Open Geospatial Consortium

Date: 2011-12-19

Reference number of this document: OGC 11-093r2

http://www.opengis.net/doc/ows8-aviation-architecture

Category: Public Engineering Report

Editor: Johannes Echterhoff

OGC® OWS-8 Aviation Architecture Engineering Report

Warning

This document is not an OGC Standard. This document is an OGC Public Engineering Report created as a deliverable in an OGC Interoperability Initiative and is not an official position of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, any OGC Engineering Report should not be referenced as required or mandatory technology in procurements.

Document type:	OpenGIS® Engineering Report
Document subtype:	NA
Document stage:	Approved for public release
Document language:	English

Preface

This document is a deliverable of the OGC Web Services (OWS) Initiative - Phase 8 (OWS-8). It describes the architecture that was implemented in the OWS-8 Aviation thread.

This document is a deliverable for the OGC Web Services 8 (OWS-8) testbed activity. OWS testbeds are part of OGC's Interoperability Program, a global, hands-on and collaborative prototyping program designed to rapidly develop, test and deliver proven candidate standards or revisions to existing standards into OGC's Standards Program, where they are formalized for public release. In OGC's Interoperability Initiatives, international teams of technology providers work together to solve specific geoprocessing interoperability problems posed by the Initiative's sponsoring organizations. OGC Interoperability Initiatives include test beds, pilot projects, interoperability experiments and interoperability support services - all designed to encourage rapid development, testing, validation and adoption of OGC standards.

The OWS-8 sponsors are organizations seeking open standards for their interoperability requirements. After analyzing their requirements, the OGC Interoperability Team recommend to the sponsors that the content of the OWS-8 initiative be organized around the following threads:

* Observation Fusion

* Geosynchronization (Gsync)

* Cross-Community Interoperability (CCI)

* Aviation

More information about the OWS-8 testbed can be found at:

http://www.opengeospatial.org/standards/requests/74

OGC Document [11-139] “OWS-8 Summary Report” provides a summary of the OWS-8 testbed and is available for download:

https://portal.opengeospatial.org/files/?artifact_id=46176

License Agreement

Permission is hereby granted by the Open Geospatial Consortium, Inc. ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD.

THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER'S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Contents

Page

1 Introduction 1
1.1 Scope 1
1.2 Document contributor contact points 2
1.3 Revision history 2
1.4 Future work 3
1.5 Foreword 5
2 References 6
3 Terms and definitions 7
4 Abbreviated terms 8
5 OWS-8 Aviation Architecture - Overview 11
6 Workflows 12
7 Component Descriptions 15
7.1 AIXM 5.1 WFS-T 15
7.1.1 Snowflake 15
7.1.1.1 Components Overview 15
7.1.1.2 Component functionality 16
7.1.1.3 Data available via the components 17
7.1.1.4 Accomplishments 17
7.1.1.5 Challenges 17
7.1.2 Comsoft 18
7.1.2.1 Overview 18
7.1.2.2 Purpose in OWS-8 18
7.1.2.3 WFS 2.0 conformance 19
7.1.2.4 New and specific functionality and other contributions to OWS-8 19
7.1.2.5 Digital NOTAM generation 21
7.1.2.6 Metadata support 21
7.1.3 Luciad 21
7.1.4 WFS Service Capabilities - Summary 21
7.2 WXXM 1.1 WCS 2.0 24
7.2.1 GMU 24
7.3 FPS 24
7.3.1 Carmenta 24
7.3.2 Envitia 25
7.3.2.1 Introduction 25
7.3.2.2 Standard Functionality 25
7.3.2.3 Specific/new Functionality 25
7.3.2.4 Available Data 25
7.3.2.5 Component Configuration 26

7.3.2.6 Challenges Faced 26
7.3.2.7 OWS-8 Accomplishments 26
7.3.3 Luciad 27
7.4 Event Service 28
7.4.1 UM-IfGI 28
7.4.1.1 Interfaces 28
7.4.1.2 Changes to the Event Service Implementation 29
7.4.2 IDS 30
7.4.2.1 Architecture 30
7.4.2.2 Services Description 31
7.4.2.3 Configuration 32
7.5 Registry Service 32
7.5.1 Galdos 32
7.5.1.1 Key features 33
7.6 Aviation Client 33
7.6.1 Luciad 33
7.6.1.1 Challenges & Accomplishments 37
7.6.2 Frequentis 38
7.7 WXXM Client 41
7.7.1 Atmosphere 41
7.8 AIXM 5.1 Validation Tools 42
7.8.1 Lisasoft 42
7.8.1.1 Schematron Rules 43
7.8.1.2 Use in OWS-8 43
7.9 AIXM Performance Assessment Tools 44
7.9.1 AtoS 44
7.9.1.1 The EXI-TTFMS framework 44
7.9.1.2 Test cases 45
7.9.1.3 Results 45
7.9.1.4 Future work 46
7.10 Access Control System 46
7.10.1 TUM 46

8 Access Control System within the OWS-8 Aviation Architecture 48
8.1 Service-oriented Security Architecture 48
8.2 Initiation of the Access Control Process 48
8.3 Architecture of XACML based Access Control Systems 50

9 Aviation Event Architecture 53
9.1 Encoding of Aviation Events 53
9.1.1 Digital NOTAM 53
9.1.1.1 Temporality and Uniqueness 53
9.1.1.2 Spatial Extent 54
9.2 Web Service Notification and SOAP 54
9.3 Eventing Components and Dataflow 55
9.3.1 Determining Data of Interest 56

9.3.2 Data Provision 57
9.3.3 Data Processing 58
9.4 Dynamic Filtering 58
9.4.1 Creation of a Dynamic Spatial Filters 59
9.4.2 Provision of Aircraft Position Data 62
9.4.3 Processing of Position Data 62
9.5 Observed Issues and Drawbacks 62
9.5.1 Dealing with the AIXM Temporality Model 63
9.5.1.1 Event Service Conformance Classes 63
9.5.1.2 WFS Support for Dynamic Features 63
9.5.2 Interoperability between Event Services 64
9.5.3 Additional Observations 64

10 Lessons Learned 66
10.1 AIXM / Temporality Model 66
10.1.1 Clarify Snapshot Definition 66
10.1.1.1 Problem Statement and Description 66
10.1.1.2 Recommendation 68
10.1.2 Clarify Snapshot Encoding for Feature Property with Schedule 68
10.1.3 Extract - Extending the Snapshot Concept 73
10.1.3.1 Introduction 73
10.1.3.2 Use Case 74
10.1.3.3 Computation 75
10.1.3.4 Encoding 79
10.1.3.5 Summary 83
10.1.4 Reconsider Rules for Handling Changes to Multi Occurring Properties 84
10.1.4.1 Delta for multi-occurring property with schedule 84
10.1.4.2 Temporary change of multi-occurring property that overlaps a permanent change 84
10.1.4.3 Recommendation 89
10.1.5 Incorporate Extension Property Handling 90
10.1.6 Temporality Model as Standalone Specification 96
10.1.6.1 Purpose of the Temporality Model 96
10.1.6.2 Why the Temporality Model should become a Standalone Concept 96
10.1.6.3 What is needed? 97
10.2 SOAP/WSDL support in OWS 102
10.2.1 SOAP Complexity 102
10.2.2 Bootstrapping a SOAP based OWS 103
10.2.3 SOAP Version 104
10.2.4 Security 105
10.2.5 Message Size 105
10.2.6 Conclusion 106
10.3 Unit of Measure Handling in Filter Expressions 106
10.3.1 Background 106
10.3.2 Enabling automated Unit of Measure Conversion 107

10.3.3 Conclusion 109
11 Scenarios 109
12 Accomplishments 111
13 Annex A – Detailed Scenario Descriptions 113
13.1 Dispatch and Planning 113
13.2 Increasing Situational Awareness for Flight Planners, Pilots and Operation Centers 114
13.3 Probabilistic Weather in Decision Making 114

Figures

Page

Figure 1 – OWS-8 Aviation Architecture – High-Level Overview 11
Figure 2 – Common component interactions to retrieve and disseminate data 12
Figure 3 – Component interactions to portray data 13
Figure 4 – Overview of the Snowflake aviation component architecture 15
Figure 5 – Dataflow of Event Notifications 30
Figure 6 – IDS Event Service Architecture 31
Figure 7 – Styling based on availability and contamination type 35
Figure 8 – Class-based ICAO airspace styling 35
Figure 9 – Integrated data browser 36
Figure 10 - 3D visualization 36
Figure 11 – Using Domain Specific Query (DSL) expressions to perform queries with FES 2.0 filter expressions at WFS 39
Figure 12 – Retrieved EEVI data in the client front-end 39
Figure 13 – Creating selection on map with Carmenta FPS Airport layer enabled 40
Figure 14 – Guidance TAF visualizer - Pilot View 41
Figure 15 – Guidance TAF visualizer - Scientific View 42
Figure 16 – DuckHawk Testing Framework – Overview 43
Figure 17 – Candidate components for the initialization of the access control process 49
Figure 18 – Architecture of an XACML based Access Control System 51
Figure 19 – AIXMBasicMessage transporting a Digital NOTAM 53
Figure 20 – Components involved in OWS-8 Eventing 55
Figure 21 – Data flow between components of the Event Architecture 56
Figure 22 – Dynamic buffer of a flight route 59

Figure 23 – State tracking of the dynamic spatial buffer 61

Figure 24 – Snapshot with Baseline and Tempdelta 67

Figure 25 - Snapshot with Baseline, Tempdelta and new Permdelta without according Baseline update 68

Figure 26 – State of an AIXM feature defined via its timeslices and time of interest of an Extract of that feature 76

Figure 27 –State of an AIXM feature defined via its timeslices - timeslices relevant for the Extract with given time of interest are depicted in purple. 78

Figure 28 –Timeslices relevant for an Extract (depicted in purple) whose time of interest is partially outside the feature lifetime. 79

Figure 29 – Extract whose time of interest is during the feature lifetime - encoded as a list of Snapshots 80

Figure 30 – Extract whose time of interest overlaps the feature lifetime - encoded as a list of Snapshots 81

Figure 31 – Extract including temporary changes by schedule - encoded as a list of Snapshots 83

Figure 32 – Permanent change of a multi occurring feature property occurring during the valid time of a temporary change of that property 85

Figure 33 – Handling the delta overlap issue for a multi-occurring property without schedule 86

Figure 34 – Options for solving the delta overlap issue for a multi-occurring property with schedule 87

Figure 35 – Solving the delta overlap issue for a multi-occurring property with schedule via the schedule itself 88

Figure 36 – Solving the delta overlap issue for a multi-occurring property with schedule by encoding the temporary change via two Tempdeltas 89

Figure 37 – Special Use Airspace Feature extension (for AIXM 5.1) 90

Figure 38 – Special Activity Airspace extension (for AIXM 5.1) 91

Tables

Page

Table 1 – WFS Service and Operation Capabilities 21

Table 2 – WFS Filter Capabilities 22

Listings

Page

Listing 1 – XPath subscription 57

Listing 2 – FES 2.0 subscription **57**
Listing 3 – Example digital NOTAM Notification **58**
Listing 4 – Generic restrictive view **60**
Listing 5 – Update for the position of an aircraft **62**
Listing 6 – Subscribe response **64**
Listing 7 – Airspace with activation based on schedule **69**
Listing 8 – Airspace Snapshot with full schedule **71**
Listing 9 – Airspace Snapshot with single value without schedule **72**
Listing 10 – AIXM Airspace Baseline with both SUA and SAA extension elements **92**
Listing 11 – AIXM Airspace Permdelta with changed SUA extension element **94**

OGC® OWS-8 Aviation Architecture Engineering Report

1 Introduction

1.1 Scope

This OGC® document describes the architecture implemented in the OWS-8 Aviation thread, including general workflows. The document contains a summary description of the various components within the architecture. An introduction to the Access Control System is provided. Furthermore, the document describes relevant aspects of handling events and notifications. Lessons learned – for example regarding the AIXM Temporality Model – as well as scenarios and accomplishments are documented as well.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium Inc. shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

1.2 Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Name	Organization
Costantino Saponaro Luca Giallombardo	IDS
Daniel Tagesson	Carmenta
David Burggraf	Galdos
Debbie Wilson	Snowflake
Jan Hermann	TUM
Jeroen Dries	Luciad
Jérôme Jansou	AtoS
Jim Groffen	Lisasoft
Johannes Echterhoff (editor)	iGSI
Matthes Rieke	IfGI
Nadine Alameh	OGC
Rob Atkinson	CSIRO
Simon Cox	CSIRO
Simon Merrick	Envitia
Thibault Dacla	Atmosphere
Timo Thomas Ulrich Berthold	Comsoft
Yuqi Bai	GMU
Zdenek Farana	Frequentis

1.3 Revision history

Date	Release	Editor	Primary clauses modified	Description
2011-08-24	0.1	Johannes Echterhoff	all	initial word version
2011-09-09	0.2	Johannes Echterhoff	throughout	integrated a number of contributions and completed various sections
2011-09-30	1.0	Johannes Echterhoff	throughout	final version

1.4 Future work

The following items were identified for consideration in future initiatives:

- **Data validation via WPS** - The work performed on validating the Digital NOTAM Event Specification (DNES) revealed that integrating a validation tool to check DNOTAM business rules on given dnotam:Event instances into the Aviation service infrastructure can be facilitated by encapsulating the validation process in a web service.

  Development of a WPS profile for validation of XML instances against their respective schema and possibly existing business rules appears to be a valuable effort - not only for the Aviation domain but also for the general OGC community - see future work item “Invocation of validation tools” in the DNOTAM ER for a more detailed discussion of this idea

- **Testing and integration of automatic unit of measure conversion functionality** - Automated UoM conversion as described in section 10.3 would improve the filter capabilities of OGC Web Services. Especially when the UoM of a feature property is not constant, the mechanism helps to perform meaningful comparison operations. The discussion performed in OWS-8 on this topic did not cover all relevant aspects (see section 10.3 for further details). Therefore, a future activity should further develop the automatic UoM conversion mechanism, integrate it into actual OGC Web Service implementations (like WFS and Event Service that support OGC Filter Encoding Specification) and test it. Eventually, the mechanism can then be integrated in or become an extension of the OGC Filter Encoding Specification.

- **Investigate, improve and provide guidance on service bootstrapping** – During OWS-8 participants made experience with bootstrapping of SOAP based web services. Apparently there is a need for additional experience and guidance both for web service providers and clients on bootstrapping to such services. While the traditionally used OGC service bindings – HTTP GET (KVP) and HTTP POST – are sufficiently described via an OGC service’s Capabilities document, this is not the case for SOAP based web services. Here, the WSDL document provides additional information that has not been considered for an OWS Capabilities document (such as SOAP version and operation action identifiers). Section 10.2.2 explains the issue and possible solutions in more detail.

  Future work should consider testing the various options with different types of OGC web services and generic clients. Guidance should be developed regarding the way that clients can readily start interacting with a SOAP based OGC web service and also the information that needs to be contained in the metadata of that service. Ultimately, this work would help integrating and using OGC services – but also clients that want to interact with these services – in environments that apply standards and technologies from the more general IT world. Such an environment would be, for example, an Enterprise Service Bus (ESB) with OGC

services deployed on it.

Work on SOAP and WSDL for OGC Web Services has been performed in previous OGC initiatives (see OGC 08-009r1). This work should be reviewed and updated/extended to take into account IT standards such as WS-Addressing, WSDL 2.0 and the WS-I BasicProfiles.

- **Temporality Model as standalone specification and better support for time varying data in OGC/ISO standards** – The AIXM Temporality Model defines encoding as well as business rules that enable applications to keep track of time varying data, in this case of AIXM features and their properties. Refactoring the Temporality Model into a standalone specification/standard would be beneficial for the following reasons: a) enable re-use of the concepts defined by the Temporality Model in other domains that deal with time varying data, b) improved maintenance/governance of the Temporality Model itself, c) improved interoperability and usability when handling and managing time varying data via (web) services. Section 10.1.6 explains this in more detail. The ideas and suggestions presented in that section should be considered in the future. Improved usability and interoperability in model design and encoding as well as management and access of time varying information are key benefits of the suggested refactoring. Section 10.1.6 also outlines how this could be achieved by revising/extending a set of OGC and ISO standards.

- **Aviation Event Architecture**
  - **Dynamic spatial filtering** – Future work on the dynamic spatial filter should cover processing as well as (architecture) modeling and interface definition aspects. A combination of temporal filters and dynamic spatial filters is imaginable. For instance, an SAA in a near upcoming part of the flight route may be activated but the start time of the activation is three hours from now. Obviously, a pilot or dispatcher is not interested in such an Event as the airplane will have passed this airspace already three hours before the activation takes place (given that the route segment can be traversed in three hours). Missing such events because of application of dynamic filtering must be regarded from a security aspect as well; however, this mechanism can help clients to receive and act only upon those events that are of interest. Additionally, the design of the EML should be improved to reduce the current complexity. This accompanies a general review of the EML model with respect to dynamic event patterns and parameters. In general, alternative approaches to using EML could be investigated.
  - **Simplification of subscription methods** – Specific requirements from a client perspective should be included in the design of subscriptions. This includes the simplification of interfaces as a client should not deal with such complex markup languages if only a rather small and specific subset of the given functionality is used. This concerns the way that filter statements are created. Various options to improve the situation should be

discussed such as domain specific filter functions and stored subscriptions in Event Services. Work topics could cover the development of interfaces as well as general investigations on contents for such stored subscriptions.

- **Provision of aircraft position updates** – in the case that the receipt of aircraft positions using ADS-B is considered as appropriate several architectural aspects need to be taken into consideration. From an Event Service point of view receiving a vast amount of position updates from different aircrafts is problematic due to performance limitations. The event architecture could benefit from some sort of broker service for position updates. An Event Service receiving a subscription for a specific flight route (with an assigned CallSign) could then subscribe at such a broker service for position updates of the aircraft with this call sign. Thus, an Event Service only receives those position updates which are relevant for the currently registered subscriptions. In general, it may also be reasonable to hide such a broker architecture behind the façade of a possible authoritative Event Service which would provide a global interface for subscribing to DNOTAMs.
- **Conceptual work on enrichment of thin events within an Event Service** – during this testbed the developed enrichment design has been applied while processing events. What is currently missing is the conceptual integration of this feature into the overall model of the Event Service. Future work on the enrichment of thin events should consider the integration into the service metadata. A client using the Event Service would then be able to determine if it is capable of pulling additional information from a (WFS) data store, thus being able to define appropriate subscription filters. This would also imply the definition if an Event Service supports the retrieval of AIXM feature Extracts for a certain period of time (see section 10.1.3 for further details on “Extract”).

1.5 Foreword

This document is a deliverable of the OGC Web Services (OWS) Initiative - Phase 8 (OWS-8). It describes the general architecture that was implemented in the OWS-8 Aviation thread. It also contains summaries of the components developed for and used in OWS-8 Aviation. Furthermore, it documents issues, lessons learned as well as accomplishments and scenarios that were of general interest in the Aviation thread. More detailed information on specific aspects considered in OWS-8 Aviation, such as WFS usage, performance benchmarking and ICAO based portrayal can be found in dedicated Engineering Reports developed in OWS-8 (see chapter 2).

2 References

The following documents are referenced in this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

OWS-8 Engineering Reports:

- [OGC 11-061] OWS-8 AIXM Metadata Guidelines Engineering Report
- [OGC 11-072] OWS-8 Aviation - WXXM Interim Engineering Report
- [OGC 11-073] OWS-8 Aviation: Guidance for retrieving AIXM 5.1 data via an OGC WFS 2.0
- [OGC 11-086] OWS-8 Aviation Thread – Authoritative AIXM Data Source Engineering Report
- [OGC 11-089] OWS-8 Engineering Report - Guidelines for ICAO portrayal using SLD/SE
- [OGC 11-091] OWS-8 WXXM/WXXS Schema Validation Results
- [OGC 11-092] OWS-8 Report on Digital NOTAM Event Specification
- [OGC 11-097] OWS-8 AIXM Compression Performance Benchmarking ER
- [OGC 11-106] OWS-8 Digital NOTAM Refactoring Report
- [OGC 11-107] OWS-8 Domain Modeling Cookbook

Other OGC Documents:

- [OGC 08-009r1] OWS 5 SOAP/WSDL Common Engineering Report
- [OGC 10-079r3] OWS-7 Aviation Architecture ER
- [OGC 10-195] Requirements for Aviation Metadata
- [OGC 10-196r1] Guidance on the Aviation Metadata Profile
- [OGC 11-060] Use of GML in aeronautical data
- [OGC 11-055] SAA Pilot Study Engineering Report

Aviation Documents:

- Digital NOTAM Event Specification, ed. 1.0 (Proposed Release), online at http://www.aixm.aero/public/standard_page/digital_notam_specifications.html
- AIXM - Temporality Model v1.0, online at http://www.aixm.aero/public/standard_page/download.html
- AIXM - AIXM Application Schema Generation, online at http://www.aixm.aero/public/standard_page/download.html
- AIXM - UML to XML Schema Mapping v1.1, online at http://www.aixm.aero/public/standard_page/download.html

Other Documents:

- The Unified Code for Units of Measure (UCUM), online at http://aurora.regenstrief.org/~ucum/ucum.html

3 Terms and definitions

For the purposes of this report, the following terms and definitions apply.

3.1
Extract
Information on the complete status of an AIXM feature during a given time interval.

3.2
Dynamic property
Synonyms: dynamic data, time varying property, time varying data
Property of a feature type, type or data type whose value(s) and value changes are tracked according to the AIXM Temporality Model.

4 Abbreviated terms

ACARS	Aircraft Communications Addressing and Reporting System
ACS	Access Control System
ADR	Authorization Decision Request
ADS-B	Automatic Dependent Surveillance-Broadcast
AIM	Aeronautical Information Management
AIP	Aeronautical Information Publication
AIRAC	Aeronautical Information Regulation and Control
AIXM	Aeronautical Information Exchange Model
AIXM-TM	AIXM Temporality Model
AOA	ACARS over AVLC
ATC	Air Traffic Control
ATN	Aeronautical Telecommunication Network
AVLC	Aviation VHF Link Control
BBOX	Bounding Box
CEP	Complex Event Processing
COTS	Commercial Off the Shelf
CRS	Coordinate Reference System
CSW	Catalog Service for the Web
DCMI	Dublin Core Metadata Initiative
DGIWG	Digital Geographic Information Working Group
DNES	Digital NOTAM Event Specification
DNOTAM	Digital NOTAM
DSL	Domain Specific Language
ebRIM	Electronic Business Registry Information Model
EFB	Electronic Flight Bag
EML	Event Pattern Markup Language
ES	Event Service
ESB	Enterprise Service Bus
EXI	Efficient XML Interchange
FAA	Federal Aviation Administration
FES	Filter Encoding Specification

FPS	Feature Portrayal Service
GeoJSON	Geographic JavaScript Object Notation
GFM	General Feature Model
GML	Geography Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
IATA	International Air Transport Association
ICAO	International Civil Aviation Organization
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
KML	Keyhole Markup Language
METAR	Meteorological Aerodrome Report (may vary)
NAWX	North American Weather extension to WXXM
NOTAM	Notice to Airmen
OASIS	Organization for the Advancement of Structured Information Standards
OGC	Open Geospatial Consortium
OWS	OGC Web Service
PAP	Policy Administration Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
POA	Plain Old ACARS
RIA	Rich Internet Application
SAA	Special Activity Airspace
SE	Symbology Encoding
SES	Sensor Event Service
SIGMET	Significant Meteorological Information
SLD	Styled Layer Descriptor
SOA	Service Oriented Architecture
SSL	Secure Sockets Layer
SUA	Special Use Airspace
SWIM	System Wide Information Management

TAF	Terminal Aerodrome Forecast
TLS	Transport Layer Security
UCUM	Unified Code for Units of Measure
UoM	Unit of Measure
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VHF	Very High Frequency
W3C	World Wide Web Consortium
WCS	Web Coverage Service
WFS	Web Feature Service
WFS-T	WFS-Transactional
WMS	Web Map Service
WPS	Web Processing Service
WS-A	Web Services Addressing
WSDL	Web Services Description Language
WS-I	Web Services Interoperability Organization
WS-N	Web Services Notification
WXXM	Weather Information Exchange Model
XACML	Extensible Access Control Markup Language
XML	Extensible Markup Language
XPath	XML Path Language

5 OWS-8 Aviation Architecture - Overview

The OWS-8 Aviation thread architecture can be separated into three tiers (see Figure 1):

- The *Client Tier* contains the client applications.
- The *Business Process Tier* contains components that offer services on top of the Access Tier: discovery, portrayal, authentication and authorization, validation and publish/subscribe
- The *Access Tier* contains Web Feature Services serving AIXM 5.1 data as well as WFSs and Web Coverage Services serving WXXM and other data.

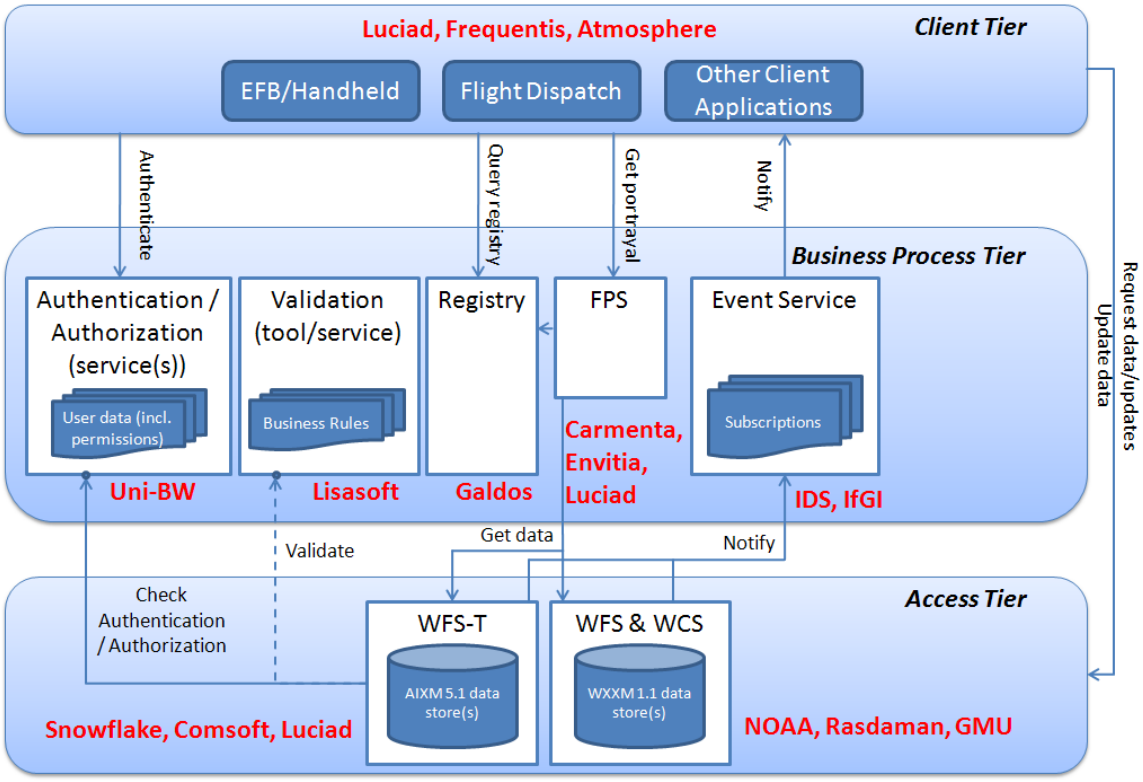

Figure 1 – OWS-8 Aviation Architecture – High-Level Overview

Figure 1 shows the links between the tiers and the general functionality that is invoked. It also shows which participants provided which components. Some components are not represented in the figure as they were not included in the service architecture, for example the AIXM Performance Assessment tool provided by AtoS. A summary description of the components is provided in chapter 7.

6 Workflows

This section provides a high-level overview of the common interactions between client and service components in the OWS-8 Aviation service infrastructure.

The following sequence diagram shows common interactions for retrieving and disseminating data that is of interest to a client.

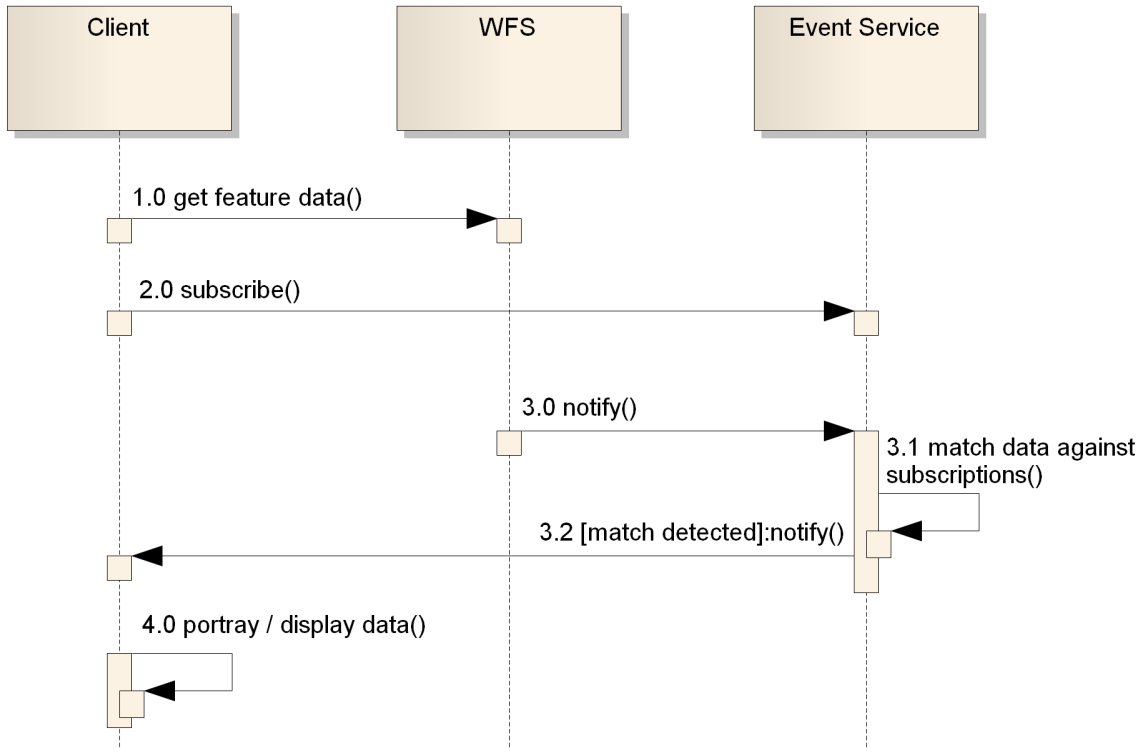

Figure 2 – Common component interactions to retrieve and disseminate data

The interactions shown in Figure 2 have the following purpose:

1.0: The client retrieves feature data from the WFS (e.g. via the GetFeature operation). The client can request general information, for example on airspaces and airports. Using specific filter criteria it can also query the WFS to identify suitable alternate/diversion airports (e.g. by searching for airports that have a passenger terminal, re-fueling facilities, a hard-surface runway of certain required minimum length etc). Weather data formatted as METARs and TAFs can also be served by and requested from a WFS.

2.0: The client creates a subscription at an Event Service to automatically be notified whenever the Event Service received new data that matches the subscriptions filter criteria. The subscription can for example be used to receive DNOTAMs for airports (e.g. the destination and alternates) and airspaces (e.g. activations). In this step, only the subscription is created and the interaction ends. Actually publishing new data is handled in a separate step – see step 3.0.

3.0: Whenever a WFS detects a relevant change in its feature data (for example that an airspace got a new activation or that a runway was closed) it generates a notification that represents the change (for example encoded as a DNOTAM) and sends it to the Event Service.
3.1: The Event Service processes the content of the notification and matches it against all subscriptions to detect matches.
3.2: If the data matches a given subscription, then the Event Service notifies the recipient defined for the subscription (here it is the client).

4.0: The client processes the data that it either retrieved from a WFS or received from an Event Service and usually displays it. Portrayal of the data is performed according to some style.

Various options exist how the portrayal of data according to styles can be achieved in the Aviation Architecture. These options are described in the following sequence diagram.

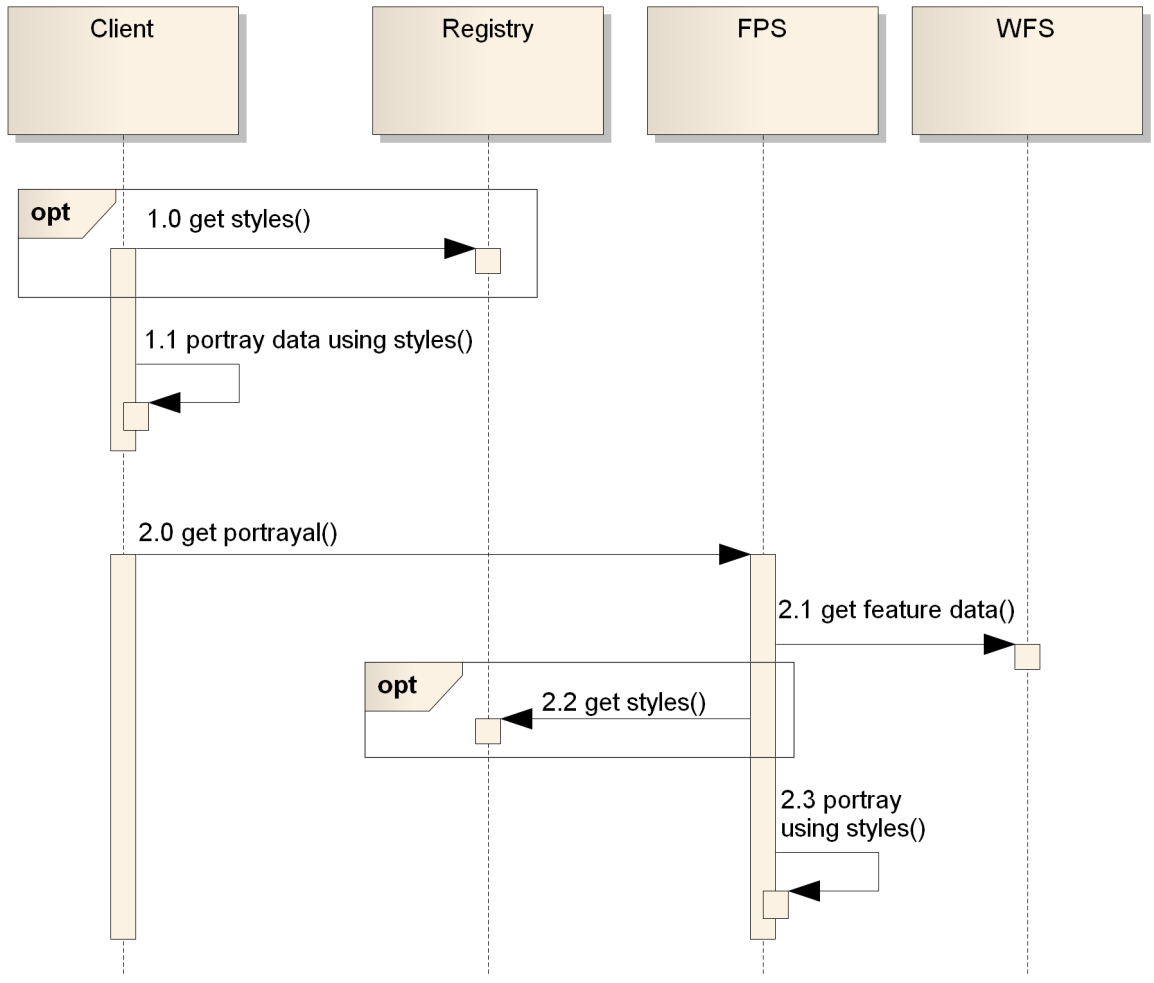

Figure 3 – Component interactions to portray data

The interactions shown in Figure 3 have the following purpose:

1.0: A client that is capable of performing style based data portrayal may first have to retrieve suitable styles from the Registry.
1.1: With style information being available, the client can then portray the data and display it.

2.0: Some clients may not be capable of creating style based portrayals themselves. These clients can use the Feature Portrayal Service to perform the job for them.
2.1: First of all, the FPS retrieves the necessary feature data from the WFS – either a WFS that is set for a pre-configured layer or a WFS chosen by the client.
2.2: If the client requested that the portrayal is performed with a certain style that is stored in a Registry, the FPS retrieves it.
2.3: Finally, the FPS portrays the data according to the styling instructions and returns the result to the client.

7 Component Descriptions

7.1 AIXM 5.1 WFS-T

7.1.1 Snowflake

7.1.1.1 Components Overview

Snowflake Software's GO Publisher commercial off-the shelf (COTS) product is comprised of a series of flexible, scalable components capable of supporting the transformation and data exchange requirements of aeronautical information systems (see Figure 4).

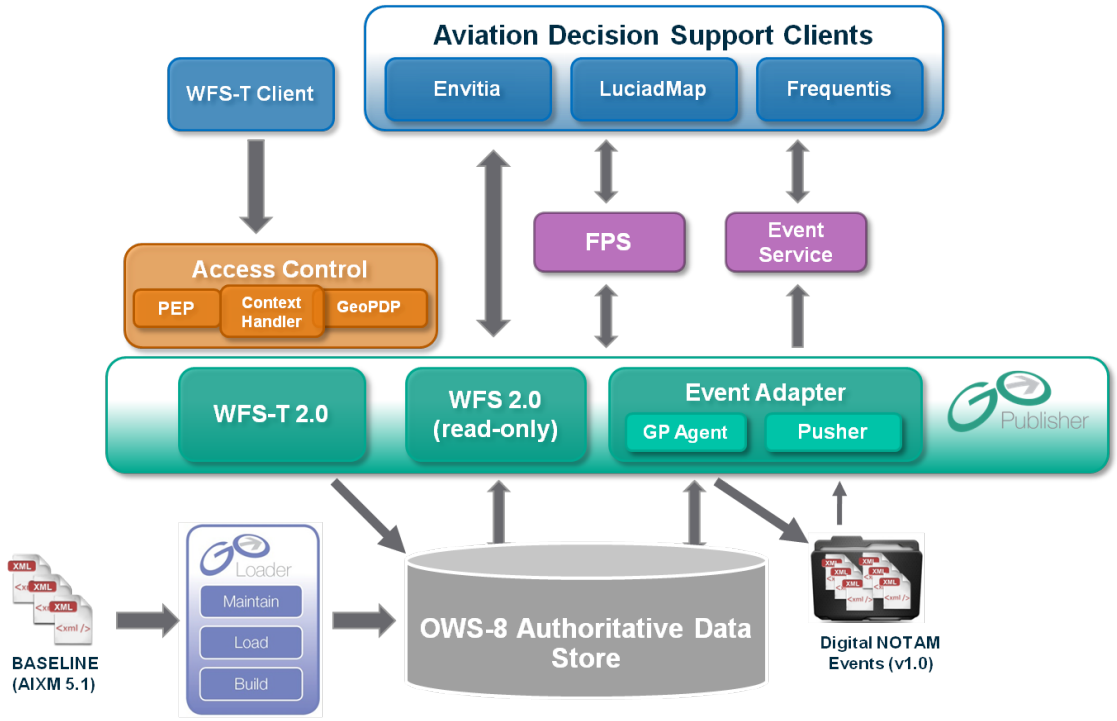

Figure 4 – Overview of the Snowflake aviation component architecture

Aeronautical data were received from a wide range of sources and integrated into a consolidated database. The consolidated database schema is an AIXM 5.1 schema storing BASELINE, TEMPDELTA and PERMDELTA timeslices. Supporting metadata were also defined based on user scenarios defined in the metadata guidelines document. The Snowflake Software component architecture consisted of three components that were used in the authoritative data source, event architecture and flight planning scenarios.

7.1.1.1.1 WFS 2.0 (read-only)

Three different instances of the read-only WFS 2.0 were established to investigate how different methods for configuring the WFS could be applied to address the key issue of retrieving features containing only those timeslices that correspond to the filter criteria contained in a request:

- **Current History WFS:** within this configuration a single feature is published representing each real world object. Each feature contains only BASELINE and TEMPDELTA timeslices that are valid now and in the future.
- **Full History WFS:** within this configuration a single feature is published representing each real world object. Each feature contains all BASELINE and TEMPDELTA timeslices that have ever been published.
- **1 Feature – 1 Timeslice WFS:** within this configuration multiple features are published that represent the real world objects. Each feature contains only one timeslice: BASELINE, TEMPDELTA or PERMDELTA that representing the state of the real world object or an event that resulted in permanent or temporary change to the feature over a specified time period.

7.1.1.1.2 WFS-T 2.0 (transactional)

A separate transactional WFS 2.0 instance was established to support the authoritative data source and event architecture threads. The WFS-T was established to support the insert of features containing TEMPDELTA timeslices representing i) SAA Activation Schedules (create, modify, approve, disapprove), ii) Navaid Unservicable, iii) Runway Closure and iv) Airport Surface Contamination.

The WFS-T was integrated into the data maintenance architecture for the consolidated database. On insert of a Feature containing a TEMPDELTA timeslice a series of processes are then triggered to auto generate additional information to publish the data via the WFS and Event Publisher in real-time.

7.1.1.1.3 Event Publisher

The Event Publisher is composed of two components:

- **GO Publisher Agent:** a server-side bulk data publishing system that generates event messages
- **Event Pusher:** this registers with one or more event services as an event source and pushes the messages generated to the event service brokers

Publication of events is triggered by the insertion of a timeslice into the database. GO Publisher Agent generates a Digital NOTAM Event which is published into a directory on the server. The Event Pusher polls this directory and on receipt of an Event message, it then pushes the message to the Event Service Brokers.

7.1.1.2 Component functionality

No new component functionality was developed for either GO Publisher WFS or the Event Publisher during OWS-8. GO Publisher WFS currently implements a large proportion of the OGC WFS 2.0 specification (Table x) and the Event Publisher continues to have sufficient functionality to connect to the IfGI and IDS Event Services.

Within OWS-8, the aim was to further evaluate the effectiveness of the basic mandatory WFS 2.0 operations and full filter encoding (FE 2.0) to retrieve AIXM 5.1 data for the various flight planning and dispatch scenarios identified.

7.1.1.3 Data available via the components

The OWS-8 consolidated database consists of data containing BASELINE and TEMPDELTA timeslices from the following sources:

- **EANS:** the EANS AIXM 5.1 was made accessible via the Comsoft WFS. Some of the feature types relevant for the scenario were harvested and loaded into the consolidated database using GO Loader.
- **Eurocontrol/LFV/ FAA Airports GIS data and AMDB Obstacles data:** these data provided for use within OWS-6 and 7 were published from the original project databases as AIXM 5.1 and loaded into the consolidated database using GO Loader.
- **Eurocontrol Digital Snowtam Trial 2011:** A global dataset of AirportHeliport AIXM 5.1 features were harvested on an ad hoc basis from the WFS established for the Digital Snowtam trial.
- **FAA NASR Subscriber data:** pre-operational data feed providing access to AIXM 5.0 SAA messages updated every 56 days inline with the AIRAC cycle. These were loaded into initially into an SAA 5.0 database then published out as AIXM 5.1 and loaded into the consolidated database using GO Loader.
- **FAA SAMS SUA Data feed:** operational data feed providing access to SAA Activation schedules (.csv) updated every minute.
- **Test Events:** a set of example test events were created to support the various scenarios that were inserted into the consolidated database via the WFS-T 2.0

7.1.1.4 Accomplishments

Several key accomplishments were developed within OWS-8 within the data maintenance and publication architecture developed by Snowflake:

- Consolidated data from multiple sources, published using different AIXM 5.1 extension schemas into a single authoritative database and provided access to these data via the WFS 2.0 and Event Publisher
- Extended the database to include some example reverse associations for AirportHeliport, Runway, Runway Element and Airspace features and tested the benefits of reverse associations for improved data retrieval
- Configured the WFS-T to support the authoritative data source and flight planning scenarios enabling users to insert TEMPDELTA timeslices (create, authorize, modify, cancel/disapprove) into the database. On insert additional processes were triggered to auto generate additional properties (i.e. sequenceNumber, correctionNumber, gml:id) for publication via Event Publisher and WFS

7.1.1.5 Challenges

Two key challenges were identified with retrieving AIXM features from the WFS 2.0.

1. **Simplifying data retrieval requiring multiple queries**

Although GO Publisher WFS implements many of the optional query parameters and filter expressions defined in the WFS specification however, it emerged that to successfully retrieve AIXM 5.1 data based on the scenarios requires the implementation of: i) Local resolve and wfs:valueOf() and ii) join query predicates. Both of these query parameters enable users to place queries that traverse properties containing xlink references to generate the response. Such queries are important in AIXM 5.1 as the relationships between features are one-directional so they should remove the requirement for reverse associations. It should also remove the requirement to submit multiple requests to obtain a result.

2. **Retrieving specific timeslices**

The WFS specification supports versioning which enable users to retrieve specific versions of a feature from within a data store, however, it does not support the ability to support the retrieval of specific timeslices from dynamic features. This is currently a serious issue; proposals for the support of timeslice retrieval are being developed in the WFS Guidance ER.

7.1.2 Comsoft

7.1.2.1 Overview

COMSOFT's Aeronautical Information Management Database (CADAS-AIM$_{DB}$) is a fully featured AIXM 5 database. It has been especially developed to natively support all concepts of AIXM 5. It is designed to serve as a base for integrating AIM products and components such as electronic AIP, Charting, NOTAM Office, or Briefing with a central Aeronautical Database.

A design principle is the interoperability with other systems. As the database is the core of any integrated AIM solution an open interface that can be used independently from any platform and programming language is one of the key features. For an optimal support of AIXM 5 CADAS-AIM$_{DB}$ provides the CAW-interface. In OWS-7, support for the emerging WFS 2.0 standard was added and further improved in OWS-8.

7.1.2.2 Purpose in OWS-8

In OWS-8, the CADAS-AIM$_{DB}$ is used as WFS data store and DNOTAM event source. As a WFS data store, it hosts static (BASELINE and PERMDELTA time slices) and dynamic data (TEMPDELTA time slices). Time slices can be retrieved from and stored to the WFS. The retrieval operations support complex filters built of logical, spatial, temporal and comparison operators.

When TEMPDELTA time slices are inserted, corresponding DNOTAM events are created and sent to interested parties by a Web Service message.

7.1.2.3 WFS 2.0 conformance

One objective of OWS-8 is to stress the WFS 2.0 standard and to apply it on AIXM 5.1 data. During this project, not all features of WFS 2.0 could be linked to a use case. Instead of reaching a maximum of WFS 2.0 conformance, COMSOFT focused on the implementation of the actual features used in the demo scenarios.

A detailed list of the supported WFS operations and capabilities can be found in Table 1.

7.1.2.4 New and specific functionality and other contributions to OWS-8

Basic WFS support for CADAS-AIM$_{DB}$ was developed in OWS-7. In OWS-8, the support was extended and new features were implemented.

7.1.2.4.1 Support for advanced GML objects

In the document "Use of GML for aviation data", version 0.3, a detailed explanation of the usage of GML in the aviation domain is given together with implementation hints. The following table lists the compliance of CADAS-AIMDB with that document:

Section	Support in CADAS-AIM$_{DB}$	Details
3.1) Use of srsName	Yes	
3.2) Use of global srsName	Partially	No inheritance from gml:boundedBy. *Developed for OWS-8*.
4) Positions	Yes	
5) Lines and Surfaces	Yes	Geodesic interpolation is always used. The "gml:Geodesic" element is not supported.
5.1) Encoding parallels	Yes	Only EPSG:4326 and CRS:84 are supported.
5.2.3) Arc by edge	Yes	*Developed for OWS-8*
5.2.4) Arc by centre point	Yes	*Developed for OWS-8*
5.2.5) Circle by center point	Yes	
6) Point references	No	The specification is not yet clear about how to implement and interpret references at geometry level as this is a deviation from the general AIXM principle of having xlink:href associations towards the feature level only. References based on gml:ids cannot take the temporality of the information into account as gml:ids are linked to a
7) Geographical border references	No	

		specific property in a specific time slice.

7.1.2.4.2 Challenges in the support for advanced GML objects

Common spatial databases do not natively support curved geometries. They support points, straight lines and complex geometries built upon them only (e.g. line strings, polygons). Curved geometries can only be stored in an approximated form by dividing them into straight lines. While implementing advanced GML objects for OWS-8, COMSOFT discovered the need for a precision parameter to spatial query operators. These findings are documented in OGC 11-073, chapter "*Precision of spatial filters*".

7.1.2.4.3 Generation and retrieval of SNAPSHOT time slices

SNAPSHOT time slices can be seen as *virtual* properties of a feature. This is because they can be calculated from BASELINE, PERMDELTA and TEMPDELTA time slices. The SNAPSHOT generation involves a complex merge process, in which time slices have to be ordered, overlaid and filtered.

CADAS-AIM$_{DB}$ already supported the retrieval and filtering of SNAPSHOT time slices through its CAW interface. In OWS-8, COMSOFT made the SNAPSHOT support available for the WFS 2.0 interface. As SNAPSHOTs play a central role in the AIXM temporality model, this work included effort in the domain of extending the WFS specification to support it, which can be found in OGC 11-073, chapter "*Configuring a WFS 2.0 to serve AIXM 5.1*". CADAS-AIM$_{DB}$ also already supports a proposed extension of the SNAPSHOT definition which is SNAPSHOTs for time periods. See sections 10.1.3 and 10.1.3.4.1 for details.

7.1.2.4.4 Advancement of the AIXM profile for WFS 2.0

COMSOFT contributed to the work on the proposed AIXM profile for WFS 2.0, which benefitted from its operational experience. This included a proposal for a new *temporal* query type as an alternative to the complex extensions needed otherwise to enable a WFS to serve AIXM 5.1 data. This alternative interface is derived from COMSOFT's CAW interface which proved its value in productive systems. Details can be found in OGC 11-073 chapter "*Use case oriented approach to time slice retrieval with WFS 2.0*".

Further conceptual work was done in the fields of

- gml:id uniqueness (see OGC 11-073 chapter "Guidelines for ensuring gml:id uniqueness")
- reverse associations (see OGC 11-073 chapter "Implications of Reverse Associations")
- excluding of optional properties (see OGC 11-073 chapter "Excluding of optional properties")

7.1.2.5 Digital NOTAM generation

CADAS-AIMDB supports the generation of DNOTAM events following the specification given in the document "Digital NOTAM Event Specification, Proposed Release, Increment 1". When TEMPDELTA time slices are inserted via the Transaction operation, DNOTAMs with digital encoding are created and sent to interested parties by a Web Service message. In OWS-8 these are event brokers that themselves categorize, forward and distribute events to registered clients on a subscriber-publisher basis.

In the current implementation, only a basic support for DNOTAMs is available. No scenario identification or validation of business rules is performed.

7.1.2.6 Metadata support

In OWS-8, support for the storage and retrieval of metadata on time slice level was added to CADAS-AIMDB.

7.1.3 Luciad

Luciad provided an OGC Web Feature Service to retrieve Special Activity Airspace (SAA) data provided by the FAA. The WFS supports filtering on properties, spatial filtering (for instance, to retrieve airspaces that potentially affect a flight route), and basic temporal filtering.

One of the challenges with serving the SAA data is the handling of the separate airspace components that are combined using constructive geometry operations, such as unions and subtractions. The Luciad WFS supports the additional processing required for exact spatial queries on such complex airspaces. This allows you to evaluate exactly whether a point lies inside or outside of a particular airspace.

The supplied data also contained links between features, for instance from an air traffic control service to the airspace for which it provides a traffic separation service. These links pose additional issues when they are encountered by client applications. This issue is avoided by the WFS by automatically converting all links to local links to features that were included with the response.

7.1.4 WFS Service Capabilities - Summary

A Web Feature Service supports a certain set of functionality, for example operations, operation parameters, feature types, filter operations as well as operands. This section provides a summary of the respective capabilities for each WFS that is introduced in the previous sections.

Table 1 – WFS Service and Operation Capabilities

		Snowflake GO Publisher WFS	**Comsoft CADAS-aimdb-WFS**	**Luciad WFS**
Service Version	1.0	✓	×	Compliancy tested

		Snowflake GO Publisher WFS	Comsoft CADAS-aimdb-WFS	Luciad WFS
	1.1	✓	×	Compliancy tested
	2.0	✓	✓	✓
DCP	HTTP GET	✓	×	✓
	HTTP POST	✓	✓	✓
WSDL		✓	✓	
Operations	GetCapabilities	✓	✓	✓
	DescribeFeatureType	✓	✓	✓
	GetFeature*	✓	✓	✓
	GetPropertyValue*	✓	×	×
	ListStoredQueries	✓	×	×
	DescribeStoredQueries	✓	×	×
	CreateStoredQueries	✓	×	×
	DropStoredQueries	✓	×	×
	Transaction	✓	✓	✓
	GetFeatureWithLock*	×	×	✓
	LockFeature	×	×	✓
Operation Constraints	Implements Basic WFS	✓	✓	✓
	Implements Transactional WFS	×	✓	✓
	Implements Locking WFS	×	×	✓
	KVP Encoding	✓	×	✓
	XML Encoding	✓	×	✓
	SOAP Encoding	✓	✓	×
	Implements Inheritance	×	×	×
	Implements Remote Resolve	×	×	×
	Implements Result Paging	×	×	×
	Implements Standard Joins	×	×	×
	Implements Spatial Joins	×	×	×
	Implements Temporal Joins	×	×	×
	Implements Feature Versioning	×	×	×
	Manage Stored Queries	✓	×	×

* None of the WFS components support the “resolve” parameter specified in clause 7.6.4 of OGC 09-025r1 / ISO/DIS 19142

Table 2 – WFS Filter Capabilities

	Snowflake GO Publisher WFS	Comsoft CADAS-aimdb-WFS	Luciad WFS

		Snowflake GO Publisher WFS	**Comsoft CADAS-aimdb-WFS**	**Luciad WFS**
Conformance Constraints	Ad Hoc Query	✓	✓	✓
	Extended Operators	×	×	×
	Functions	×	×	✓
	Minimum Spatial Filter	✓	✓	✓
	Min Standard Filter	✓	✓	✓
	Min Temporal Filter	✓	✓	✓
	Query	✓	✓	✓
	Sorting	×	×	✓
	Spatial Filter	✓	✓	✓
	Standard Filter	✓	✓	✓
	Temporal Filter	✓	✓	✓
	Version Navigation	×	×	×
ID Capabilities	ResourceId	✓	×	✓
Logical Operators		✓ (all)	✓ (all)	✓ (all)
Comparison Operators		✓ (all)	✓ (except for PropertyIsBetween)	✓
Spatial Capabilities	Geometry Operands	gml:Envelope gml:Point gml:LineString gml:Polygon gml:Arc gml:Circle	gml:Envelope gml:Point gml:LineString gml:GeodesicString gml:Curve gml:Polygon gml:Surface gml:ArcString gml:Arc gml:ArcByCenterPoint gml:CircleByCenterPoint	gml:Envelope gml:Point gml:LineString gml:GeodesicString gml:Curve gml:Polygon gml:Surface gml:ArcString gml:Arc gml:ArcByCenterPoint gml:CircleByCenterPoint gml:Multi* gml:ArcByBulge
	BBOX	✓	✓	✓
	Equals	✓	×	✓
	Disjoint	✓	×	✓
	Intersects	✓	×	✓
	Touches	✓	×	✓
	Crosses	✓	×	✓
	Within	✓	✓	✓
	Contains	✓	×	✓

		Snowflake GO Publisher WFS	Comsoft CADAS-aimdb-WFS	Luciad WFS
	Overlaps	✓	×	✓
	Beyond	✓	×	✓
	DWithin	✓	✓	✓
Temporal Capabilities	Temporal Operand	gml:TimePeriod	gml:TimePeriod gml:TimeInstant	gml:TimePeriod gml:TimeInstant
	Temporal Operators	✓ (all)	✓ (all)	✓ (all)

7.2 WXXM 1.1 WCS 2.0

7.2.1 GMU

The NOAA GOES Wind Products WCS for OWS-8 Aviation is providing access to the NOAA GOES satellite derived high density Wind products: Cloud Drift Wind and Water Vapor Wind.

These products consist of point-based measurements about the air temperature, air pressure, wind direction and wind speed, along with the exact geo-location where the measurement was performed.

This WCS exposes GetCapabilities, DescribeCoverage and GetCoverage operations. For the GetCoverage operation, users may define the bounding box and temporal range that is of interest. The range subset on a specific field can also be defined. There are two formats that this WCS can return: WXXM and KML.

For more detailed information about this WCS, go to:
http://geobrain.laits.gmu.edu/ows8/aviationWCS.html

7.3 FPS

7.3.1 Carmenta

The Carmenta FPS is built on Carmenta Server, which is an OGC-compliant geoserver implementing WMS, WFS and CSW interfaces. In OWS-8 it is used as an FPS for portrayal of ICAO symbology using SLD/SE.

Within OWS-8 support for AIXM 5.1 has been implemented by building a generic XML reader identifying GML-features.

Within OWS-8 Carmenta has successfully implemented support for AIXM and used it for portrayal of aeronautical features according to ICAO standards.

7.3.2 Envitia

7.3.2.1 Introduction

The Envitia FPS comprises an implementation of the WMS specification. Versions 1.0.0, 1.1.1 and 1.3.0 are supported. The FPS functionality is provided by a plugin to the Envitia Web Services WMS component. The purpose of this Envitia FPS is to support the OWS-8 Aviation demonstration participants in creating their demonstrations.

The available data and capabilities of the component can be explored by visiting the Envitia Services website[1] and following the Services link (top right corner) to the FAA Demonstrator[2]. This demonstrator has been configured as the OWS-8 Aviation Demo portal. The portal is best viewed in Mozilla Firefox.

7.3.2.2 Standard Functionality

Interaction with this component is exclusively via KVP-encoded HTTP GET requests. The FPS connects to the available WFS components to obtain AIXM data. The data is then rendered using an FPS client specified SLD. The SLDs are stored in a portrayal registry.

7.3.2.3 Specific/new Functionality

In addition to the standard WMS capabilities, the Envitia FPS component also allows clients to request terrain and airspace profiles or “cross sections” as they are named in the Envitia Services portal. This can be seen by browsing to the OWS-8 Aviation Demo portal. Simply open any of the Common Operating Pictures available, select the Cross Section tab and then use the XZ tool to select a start and end point for the profile query.

No new FPS functionality was added to Envitia’s components as part of this project.

7.3.2.4 Available Data

Snowflake, Luciad and COMSOFT WFS have been consumed and are available as rendered WMS layers from the FPS when a valid SLD is specified in the WMS request.

The data has been filtered loosely to the areas of interest, namely Estonia and Hawaii. Also available are supporting background raster images for these areas. Available imagery can be discovered by browsing the Envitia OWS-8 Aviation Demo portal.

---

[1] http://services.envitia.com

[2] http://services.envitia.com/FAASAA

Feature types available from the FPS are:

- Airspaces
- Airports / Heliports
- Navaids
- Runways
- Routes
- Taxiways

7.3.2.5 Component Configuration

Envitia's ChartLink application is the nucleus of the system. It is running on the server and is accessed via the Envitia Web Services WMS component in order to service requests.

The data from the source WFSs provided by Luciad, COMSOFT and Snowflake has been pre-processed and cached.

7.3.2.6 Challenges Faced

One of the main challenges faced with Envitia's approach was matching up third party SLDs with the internal storage format of the cached data in ChartLink. ChartLink provides the images required in servicing FPS/WMS requests and is therefore responsible for rendering using SLDs. Its internal storage method is not synonymous with the GML and XPath aspects related to the usage of the SLD and WFS standards.

Due to the complexity of some of the portrayal issues, and to the lack of support for some of the optional WFS capabilities, such as XPath resolution and join predicates, most of them were not possible to display in the demonstration portal.

Other issues identified have been articulated in the Aviation Portrayal ER (see OGC 11-089).

7.3.2.7 OWS-8 Accomplishments

A number of the more complex portrayal issues were explored with the use of this component. This supported the significant contribution to the Aviation Portrayal Engineering Report from Envitia.

One of the benefits to Envitia's approach of caching the WFS data was that it permits Envitia control over the data itself, which enabled modification of key attribute values. This is instrumental in being able to ensure that the different symbols and symbology challenges can be exercised. As a simple example, this has meant that Envitia can ensure that there are runways with UNSERVICABLE status within the areas of interest.

7.3.3 Luciad

Luciad provides an OGC Feature Portrayal Service (FPS) to portray GML data. An FPS service takes away the burden of parsing and visualizing raw GML data served by a WFS. It remotely loads and renders data, and returns the result as an image that can be shown by any modern web browser.

The FPS was provided as an in-kind contribution because it entirely relied on standards and did not need any code specific to the project. In fact, the same setup is also used in other OGC projects.

Using an FPS in an application consists of the following high-level steps:

1. Determine the WFS server that is to be contacted by the FPS
2. Construct a query for the WFS server to select the required data
3. Create a Symbology Encoding (SE) feature type style to render the data from the WFS server
4. Wrap the WFS query together with the SE style in a request and send it to the FPS

The major drawback of this scenario is that it does not allow for caching of the data on the server, mainly because it cannot be predicted what data will be requested by the user. To circumvent this, the FPS can also offer layers, much like a traditional WMS, and allow dynamic styling of those layers. This is also supported by our FPS component.

One other possible improvement would be an FPS that shares a data store with a WFS. This would allow advanced querying and styling capabilities, but would not support interaction with multiple WFS services.

Another useful enhancement would be support for vector-based output formats, such as KML and GeoJSON. This could also benefit map display performance, as the client application would no longer need to request new images every time a user zooms or pans.

Technical specifications

The FPS was built using the OGC-compliant web service components provided by Luciad. The most important features include:

- Support for OGC WMS/FPS versions 1.1.1 and 1.3.0, including GET and POST requests
- Support for OGC WFS versions 1.0, 1.1 and 2.0, including SOAP requests
- Support for OGC SLD/SE versions 1.0 and 1.1
- Support for any kind of GML application schema, such as WXXM and AIXM.
- Full support to work with AIXM 5.0 / 5.1, including the temporality model

7.4 Event Service

7.4.1 UM-IfGI

The implementation of the Event Service (ES) provided by the Institute for Geoinformatics is based on the OGC Sensor Event Service discussion paper (OGC 08-133). The source code is maintained and provided in collaboration with the 52°North Initiative for Geospatial Open Source Software GmbH. Following the Publish/Subscribe paradigm of OASIS' Web Service Notification (WS-N) family of standards it acts as a notification broker. The IfGI ES has been used in aviation threads of former OGC initiatives such as OWS-6, OWS-7 and the FAA SAA Dissemination Pilot.

7.4.1.1 Interfaces

The ES consists of three endpoints. The PublisherRegistrationManager can be used to register data providers at the ES instance but is not used in this testbed and hence not described.

Endpoint	*Available Methods*	*URL*
SesPortType – broker endpoint	GetCapabilities, Notify, Subscribe, GetCurrentMessage	http://v-tml.uni-muenster.de:8080/EventService/services/SesPortType
SubscriptionManager	Unsubscribe, PauseSubscription, RenewSubscription	http://v-tml.uni-muenster.de:8080/EventService/services/SubscriptionManagerContextPath

To enable SOAP bootstrapping both endpoints provide a WSDL description (see section 10.2.2) using HTTP Get method (<endpoint-url>?wsdl).

Besides the GetCapabilities method, all available methods are defined by OASIS WS-N. The methods used in this testbed are described in the following table.

Method	*Description*	*involved OWS-8 Component*
Notify	Used to push a NotificationMessage to the ES. The contents are Digital NOTAM Events or position update messages of an aircraft (sent by clients for demo purposes) and separated into different topics (NOTAM-Topic, AircraftPosition-Topic). This method does not use request-response communication, as notifications are pushed to the ES.	WFS providers, Clients (for demo purposes)
Subscribe	Used to define a subset of all incoming notifications. A subset can be defined using Topic filters, XPath Expressions and FES 2.0 filters. A SubscriptionReference is returned to the client, enabling the management of it. A Subscription can have an InitialTermination time (encoded as a period or a date time) to define its lifetime.	Clients

Unsubscribe	Used to cancel a Subscription (e.g. when the flight has arrived).	Clients

7.4.1.2 Changes to the Event Service Implementation

Compared to the ES used in the latest OGC testbeds some major improvements have been made during this testbed. All the subsequently described features are demonstrated in the OWS-8 Aviation scenarios including the use of digital NOTAM event encoding and enrichment of these, position updates of the aircraft, multiple concurrent subscriptions, subscription lifetime handling and use of the Unsubscribe operation.

7.4.1.2.1 Enrichment of Thin Events

To enable filtering on all properties of a Digital NOTAM with regard to the Temporality Model of AIXM the thin Events received by the ES need to be enriched (see section 9.5.1.1). The approach is one of the major changes to the ES compared to the former testbeds and is described in the following; see also Figure 5.

In a first step the Event is processed and the necessary information (gml:identifier, type of AIXM feature) is pushed to a dedicated enrichment component. As the ES does not always know where the data originated the enrichment component iterates over the registered WFS instances and requests the feature using a wfs:GetFeature request with the according feature type and gml:identifier in the fes:Filter. This iteration is stopped when the feature has been successfully retrieved to avoid redundancy. The response is then parsed and the original event is enriched with the additional information. The original Digital NOTAM stays untouched and is forwarded to the clients in the same format as received.

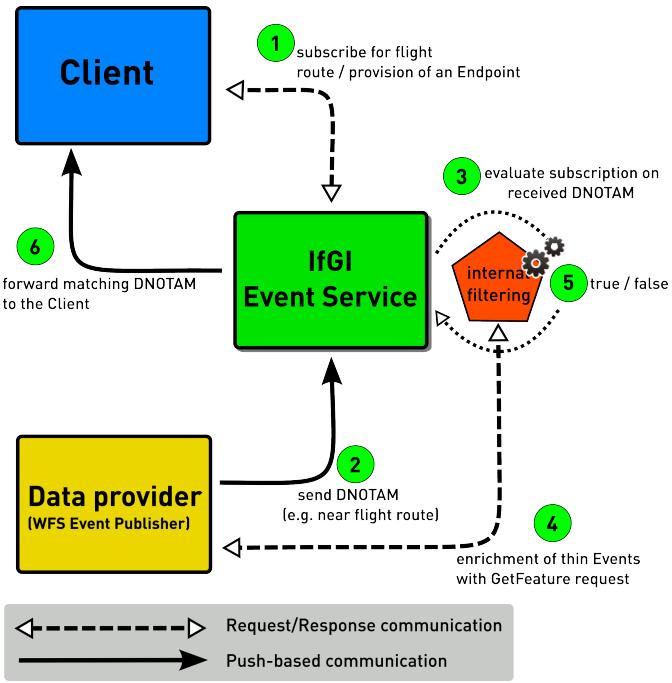

Figure 5 – Dataflow of Event Notifications

7.4.1.2.2 Dynamic Subscriptions

The dynamic filter functionality as described in section 9.4 has been implemented prototypically within this testbed. Besides the enrichment of events, this is another major change in the ES compared to OWS-7. Using the EML patterns the ES recognizes the subscription as a dynamic one and takes - in addition to the digital NOTAMs - the AircraftPositionUpdates into consideration for internal filtering. The internal representation of the subscription is continuously updated according to the retrieved positions. Thus, only those events matching the remaining flight route are forwarded to the clients.

7.4.2 IDS

The Event Service is provided by IDS Ingegneria dei Sistemi S.p.A. It is being developed in the context of OWS-8. The IDS Event Service allows distributing AIXM messages to interested clients. Clients can filter messages by message topic or by geometry. In addition to a push mode, the IDS Event Service also offers a pull mode, so that clients can retrieve messages even if not being continuously connected to the network.

7.4.2.1 Architecture

The IDS Event Service implements the OASIS WS-Brokered Notification specification. The entities involved are as follows:

- ✓ *Publisher*: A Publisher is an entity that produces Notifications (AIXM messages in this scenario) and sends it to a Broker. A Publisher can specify on what topics it wishes to publish a notification.

- ✓ *Broker*: The Broker is the entity in charge of disseminating notifications on behalf of Publishers to Consumers. Its role is routing Notifications (created by Publishers) to Consumers who own a matching subscription.
- ✓ *Consumer*: A Consumer is an endpoint designated to receive Notifications produced by a Publisher, delivered by a Broker as a result of a subscription. A Subscription represents the relationship between a Consumer and the Broker, including any filtering parameters such as Topic and spatial filter expressions.

All communication with the IDS Event Service is done via SOAP messages over HTTP. The IDS Event Service architecture is described in Figure 6.

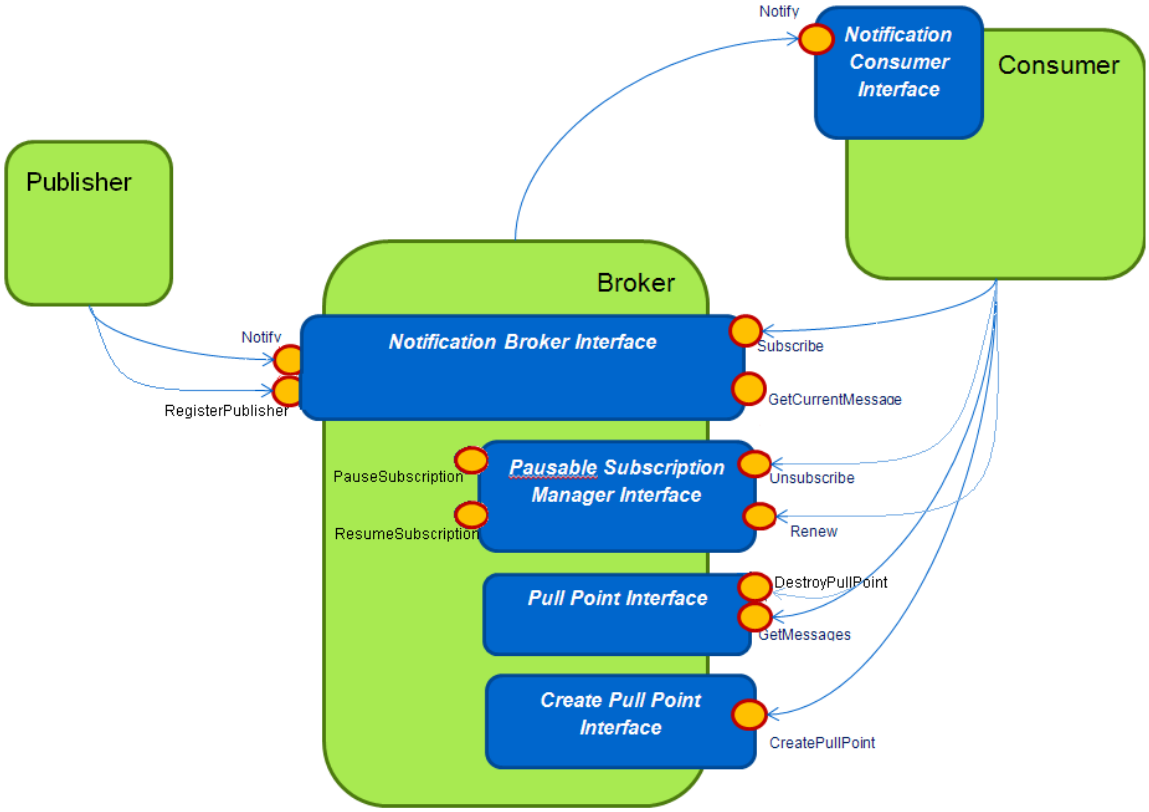

Figure 6 – IDS Event Service Architecture

In the OWS8 scenario, the IDS Event Service implements a Notification Broker; publishers are AIXM message producers; consumers are the client applications.

7.4.2.2 Services Description

The operations offered by the IDS Event Service are:

Notify: This is the basic method to send notifications (e.g. digital NOTAMs) from a notification producer to the Event Service and from the Event Service to a Notification Consumer (e.g. the clients). No response is expected from the Event Service and Notification Consumer upon receipt of this message.

Subscribe: Using this method a Notification Consumer can express its interest to receive a subset of notifications. Such a subscription may include a filter statement to specify the subset of notifications of interest. The implemented filters are by Topic and OGC Filter expression.

GetCurrentMessage: This method returns the last message that was posted to a specific topic.

RegisterPublisher/DestroyRegistration: These methods are used to register/unregister a notification producer at the Event Service. These methods are not mandatory to be used in the OWS-8 scenario, so these functionalities are not implemented in the IDS Event Services.

Renew: This method allows changing the subscription's termination time in advance.

Unsubscribe: This method cancels a specific subscription.

PauseSubscription/ResumeSubscription: These methods allow a Notification Consumer to suspend/resume its interest to receive notifications.

CreatePullPoint/DestroyPullPoint: These methods allow a Notification Consumer to create/cancel a PullPoint resource. A PullPoint is an endpoint that accumulates Notification Messages and allows a Notification Consumer to retrieve accumulated Notification Messages.

GetMessages: This method is used by the Notification Consumer to retrieve accumulated Notification Messages in the PullPoint resource previously created.

7.4.2.3 Configuration

The IDS Event Service is provided as a web application deployable under a Web Server (it is tested under Apache Tomcat). It needs a data base to store incoming messages and to save a report about sent messages result.

7.5 Registry Service

7.5.1 Galdos

Galdos INdicio™ is a Web Registry Service (WRS) that implements the Open Geospatial Consortium (OGC) ebRIM profile of the Catalogue Service for the Web, 2.0.2 specification (CSW-ebRIM 1.0.1, OGC doc 07-110r4). Galdos refers to this service as WRS for shorthand notation. The OGC Catalogue Service (OGC document 07-006r1) is an abstract catalogue standard that defines the basic notion of Record and has multiple profiles.

The INdicio Web Registry was used in OWS-8 for hosting Styling information encoded using the OGC Styled Layer Descriptor (SLD) standard that are used mainly by component Feature Portrayal Services (FPS) in OWS-8, namely by the following participants: Carmenta, Luciad, Envitia. The participants used Insert/Update transactions

to create User accounts and to publish styling resources. GetRecord was then used to retrieve the resources by users and the FPS components. No issues were encountered during the OWS-8 initiative.

Unlike conventional geographic catalogues, INdicio™ is highly configurable and can be readily deployed to manage a wide variety of objects.

7.5.1.1 Key features

- Open standards-based
- Easy installation and configuration
- Easy management and monitoring capabilities
- Secure and flexible interface
- Supports multilingual application domains
- Support for large transactions

INdicio™ ships with a CSW-ebRIM Basic Extension Package which provides a variety of useful objects for geospatial applications including:

- Services taxonomy (source: ISO 19119 “Geographic information – Services”)
- Country codes (source: ISO 3166-1 “Codes for the representation of names of countries and their subdivisions – Part 1: Country codes”)
- Geographical regions (source: UN Statistics Division)
- Feature codes (source: DGIWG FDD)
- Property categories based on Dublin Core (source: DCMI metadata terms)

INdicio™ is inherently extensible and can be adapted to multiple application requirements by using OGC CSW layered on top of ebRIM. Galdos Systems Inc. is constantly developing new extension packages, such as the CRS Extension Package, as well as custom extension packages on request.

- The Flexible and secure query interface:
- Supports spatial queries using GML 3.1.1
- Supports XACML specification
- Supports role-based access control
- Full text search of XML resources

7.6 Aviation Client

7.6.1 Luciad

Luciad contributed a client application to access aviation data through the various OGC services used in the project. The application is built with the LuciadMap software suite, offering a number of OGC-standards based components and an application framework for rapid application development. The following sections highlight the features of the client application relevant for the project.

The application provides its users with flexible means for efficient queries to WFS, FPS, and WCS services for aeronautical and weather data. These queries can be based on a geographical region and/or a time interval. Additionally, an interface to the Event Service guarantees that the user is kept informed of any updates.

The most important feature is the ability to support flight planning and dispatch operations through the use of aeronautical data in the AIXM format and weather data in the WXXM format, both delivered by WFS services. Both data types contain time dependent information that can be used by the client to adapt the visualization to the relevant time. This increases the situational awareness, either in-flight or during pre flight planning.

One of our focus points in the project was the visualization of AIXM and WXXM data using the OGC Symbology Encoding (SE) standard. This resulted in a change request for SE which can be considered an absolute necessity to make the use of SE for complex data models feasible. By implementing the proposed changes, we were able to show some promising results. Figure 7 shows an airport layout with runways and aprons respectively styled according to their availability and their contamination condition. Figure 8 shows airspace styling according to the airspace class, corresponding to the ICAO Annex 4 airspace styling guidelines.

In this respect we also had a close look at the work that is being done for the next SE version. This showed that SE 1.2 will certainly allow using more advanced styling options. For instance, we will be able to describe a hatching pattern for areas, without having to encode it as an image.

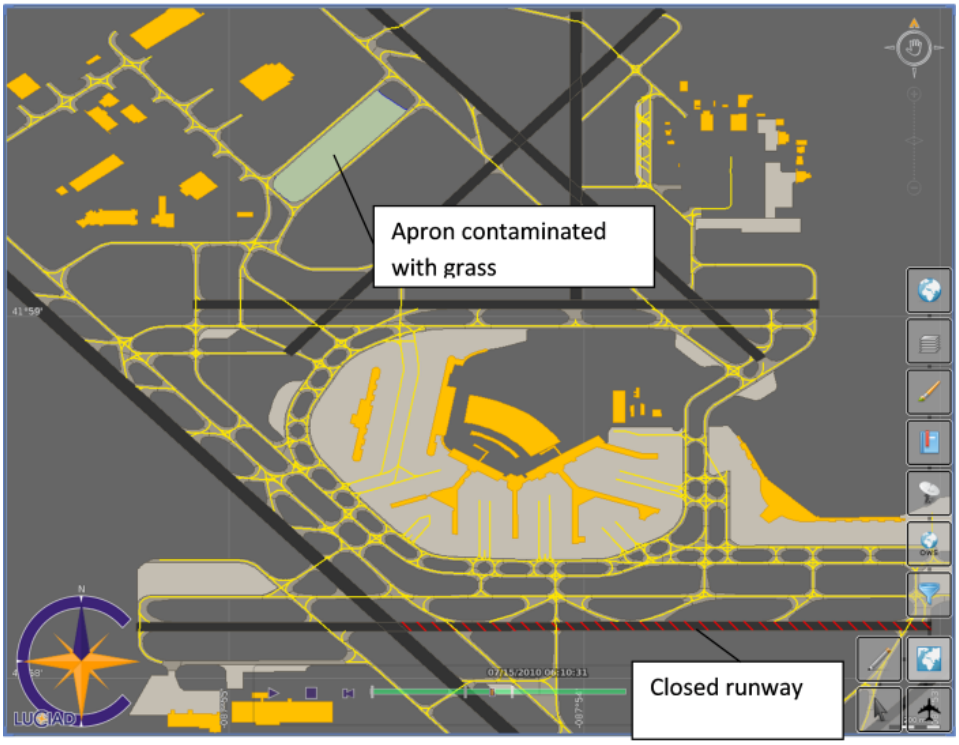

Figure 7 – Styling based on availability and contamination type

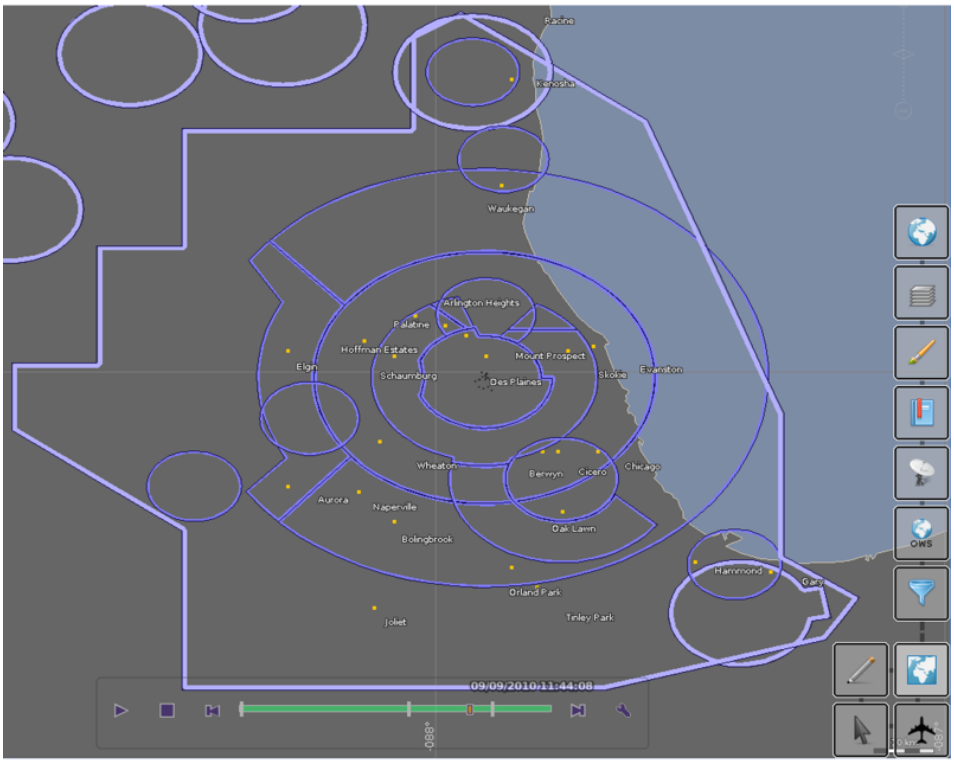

Figure 8 – Class-based ICAO airspace styling

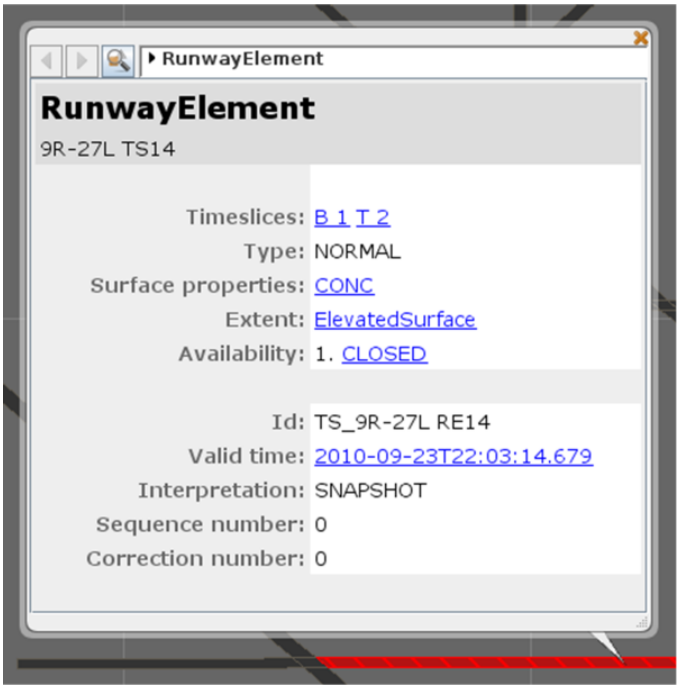

Figure 9 – Integrated data browser

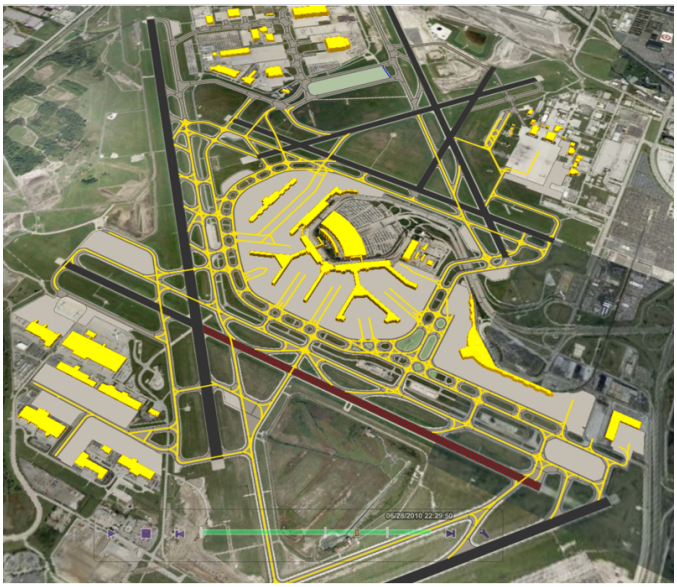

Figure 10 - 3D visualization

Other noticeable features in the application are an integrated data browser, illustrated in Figure 9, and 3D visualization capabilities, illustrated in Figure 10. The data browser enables accessing the properties of a given AIXM or WXXM feature in a way that is similar to a web browser. Users can simply follow a link from top level features to nested features or from one feature to a linked feature. Properties that are changed during the simulation of a flight are automatically updated during browsing. The 3D visualization capabilities enable the user to visualize the AIXM and WXXM data in a full 3D

environment, together with a terrain consisting of elevation data and satellite imagery. The features available for the 2D map, such as SE styling and flight simulation, are also supported in 3D, to ensure that all views are consistent.

Finally, the application also includes support for the OGC Web Coverage Service (WCS) standard, next to WFS and WMS/FPS. The WCS standard was used in the project to serve weather data in the WXXM format. Although basic interaction with WCS services was demonstrated, further investigation is needed to fully explore and evaluate the capabilities of the WCS standard in a flight planning and dispatch context.

7.6.1.1 Challenges & Accomplishments

The main challenge for a client application is to properly support interaction with the various services. This is partially caused by the use of the latest standards, which are generally not yet fully supported, but also by parts of the standard that are optional or simply not fully specified yet and thus open to interpretation. Compliancy tests and reference implementations can be very helpful tools for implementers to assess the interoperability of their products.

Retrieving the relevant data for a given flight scenario from a WFS is another challenge. Our client supports the full AIXM temporality model which allows fusion of incoming events together with pre-loaded data. This means that during the pre-flight planning phase, the dispatcher wants to assemble all AIXM data that applies to the time period of the flight by sending a query to a WFS. At the moment, the WFS 2.0 specification does not allow for such a query.

The heavy use of links between AIXM 5 features in an OGC web services environment is also an area that needs further investigation, to optimally support resolving links at the client. The WFS 2.0 standard offers some optional features that can help with this, but they are not yet in widespread use. This includes the use of wfs:valueOf and automatically including linked features in WFS responses.

Our use of the WXXM format also exposed some issues when working with WXXM data. Most important, there seem to be very little tools to ensure data quality of generated WXXM data sets. Compared to the AIXM 5 schema, WXXM allows for a lot more flexibility; this also brings a potential difficulty regarding interoperability when working with data from various sources. Schema validation is a first step, but is not a guarantee in any way that a client will be able to use the data. It also appears that some of the provided XML samples were a bit misguiding.

To visualize both AIXM and WXXM data, the client uses styles defined according to the OGC Symbology Encoding (SE) standard. This proved to be an interesting approach, but also showed that care should be taken to create styles that actually work for different applications. We have not been able to use styles created by others, sometimes due to basic errors against standards, but also due to differences in interpretations.

We also enhanced the use of SOAP by our client in an effort to increase interoperability. This was only a partial success as there seem to be some open questions on what a SOAP-based service and client should support to allow for basic interaction.

Finally, we investigated the feasibility of integrating WCS 2.0 services in our client, as there was no time left to do a complete integration. This revealed that the descriptions of coverages offered by all services were lacking the metadata that is required for the client to really understand what the server is offering. This is also a problem with older WCS services that are in use today, but can be considered an important issue.

7.6.2 Frequentis

Frequentis develops and markets communication and information solutions for safety-critical applications. It offers its control centre solutions, products and services world-wide to a broad range of customers acting in various mission-critical fields - civil air traffic management, defense, public safety, public transport and maritime. Frequentis participates in development and evaluation of various existing or emerging standards such as AIXM. Frequentis also participated in OWS-7, delivering an aviation client that was presented at several demos and the AIXM conference.

Frequentis classifies the client as EFB Class I Type B/C hybrid. It fetches data from other OWS components. The front-end is a lightweight RIA (Rich Internet Application) targeting tablet or smartphone devices such as iPad. It runs directly in a browser, hence no additional setup is needed at the user side. It is written in HTML5, jQuery and jQuery Mobile with a map component provided by OpenLayers. The user can use it with all his data even if he is disconnected from the Internet (the back-end). The back-end is written in Scala. Scala language uses JVM and it is seamlessly interoperable with Java. In order to minimize data going to the front-end it translates all data into JSON format.

The Frequentis Aviation Client can retrieve AIXM and WXXM data from WFS 2.0 compliant servers. It supports both the KVP and SOAP binding. It can use either stored queries (if available) or custom queries. In order to make querying the data easier, Frequentis has developed a very basic external DSL (Domain Specific Language) draft within OWS-8.

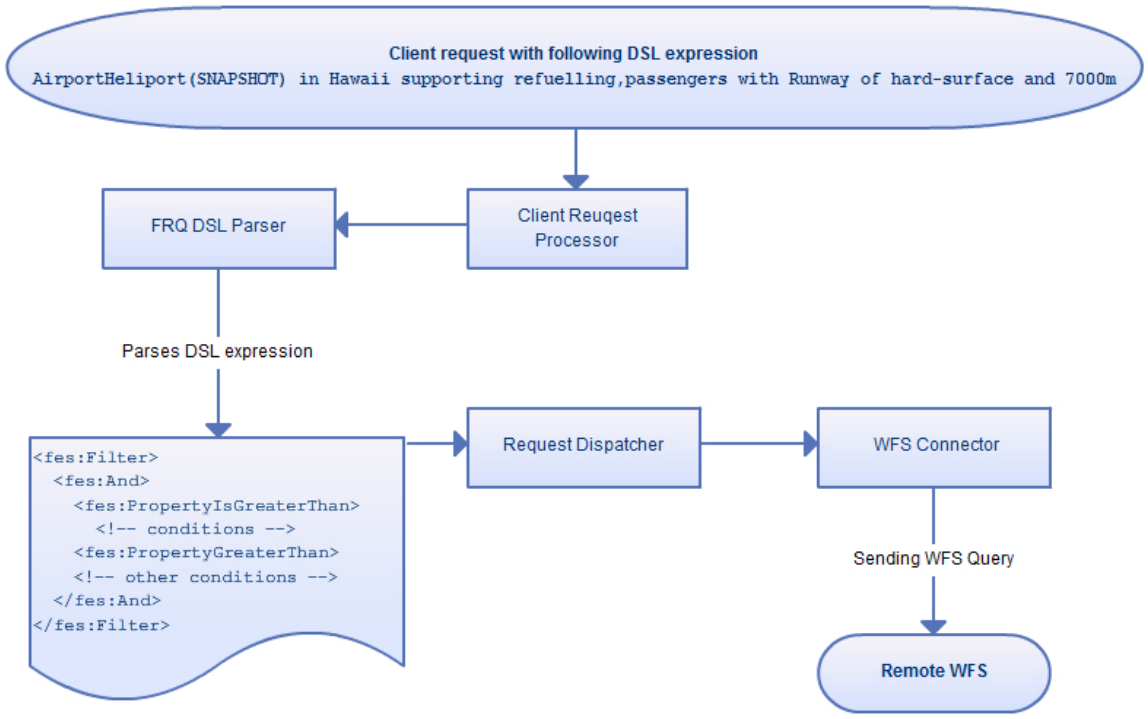

Figure 11 – Using Domain Specific Query (DSL) expressions to perform queries with FES 2.0 filter expressions at WFS

The DSL expressions resemble natural language and therefore can be easily used by flight dispatchers or pilots. Such expressions are translated into FES 2.0 (see the diagram above). The main obstacle preventing such DSL from being widely adopted is what WFS providers accept in queries. If they do not support join queries, reverse associations and/or local resolve (wfs:valueOf(..) function), it is not possible to express complex filters (e.g. filters containing references) using only one WFS query. Hence a cascade of WFS queries has to be created and the DSL parser is not suitable.

Figure 12 – Retrieved EEVI data in the client front-end

The client retrieves the basic data (departure, destination and alternate aerodromes) as SNAPSHOTs (currently supported only by COMSOFT). This data can be updated by events later. All the data retrieved by the client can be added to the *package*. Package is a concept related to the Preflight Information Briefing bulletin.

The client also integrates with Event Services. The user can subscribe for various types of events such as DNOTAM or weather data. The received events are dispatched to the user and displayed in the front-end. Such events can also affect stored data in the package and change its properties if required. The client can send dynamic filtering subscription which is very useful because it does not have to care about subscribing events along the flight route.

Spatial constraints for both retrieving WFS data and creating event subscriptions can be created via a selection tool in the map component.

The client does not have its own portraying component; it relies solely on Feature Portrayal Services. It can display layers exposed by FPS or portray data with custom SLDs. However, what is important is that FPS does not accept custom WFS data, only custom SLDs. If the client retrieves data for airport runways, it cannot send them to the FPS to be portrayed. Instead, the client calls the FPS with a custom SLD on the specified BBOX containing the runways. Then the FPS retrieves data from the WFS again and portrays it. This slightly decreases the value of the FPS as the only portraying component the client uses.

The client supports WCS coverages in KML format in a basic way – it displays them as the markers in the map.

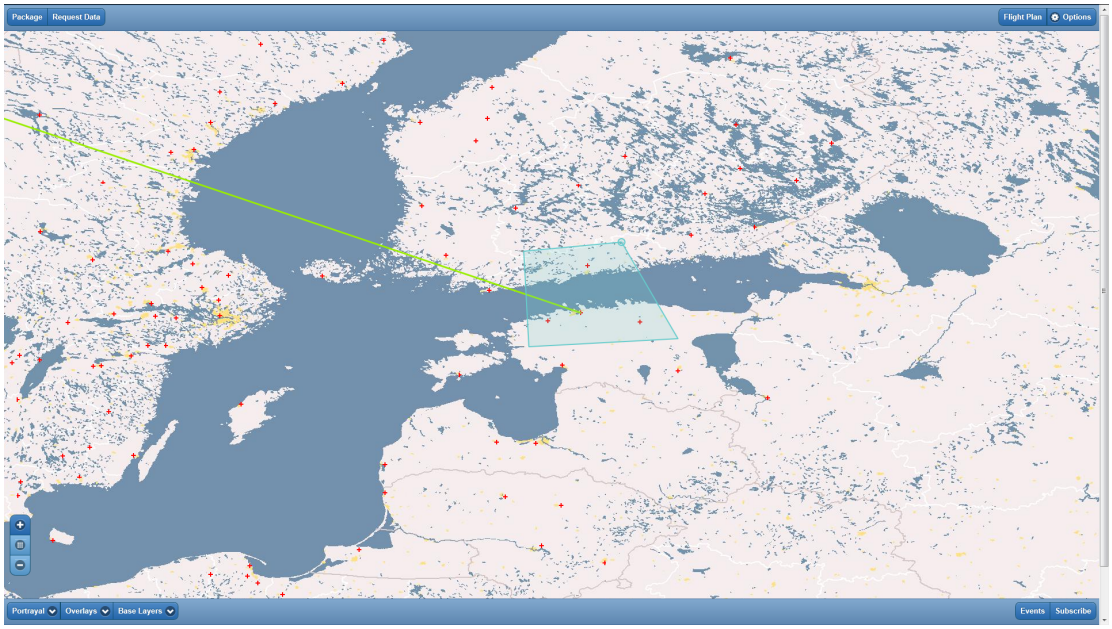

Figure 13 – Creating selection on map with Carmenta FPS Airport layer enabled

7.7 WXXM Client

7.7.1 Atmosphere

The Guidance TAF visualizer provided by ATMOSPHERE is an application able to display the probabilistic TAF in a graphical way from the Guidance TAF NAWX files provided by the NOAA WFS. NAWX is the FAA-built North American Weather extension to the WXXM standard.

The purpose of this component is three fold:

- Propose graphical methods to provide information usually presented in text form (e.g. in regular TAF files)
- Obtain feedback from pilots regarding the portrayal options and iterate with them to obtain a satisfactory portrayal policy. This feedback is aimed to demonstrate air users' interest in guidance TAF and to enhance their presentation to suit their needs/expectations.
- Demonstrate the usability of the Guidance TAF and give a tool to analyze them.

The Guidance TAF visualizer is integrated within a moving-map client, where the airports are selectable and give access to the corresponding Guidance TAF, when available.

The component implements the portrayal of the Guidance TAFs. The final goal is to achieve retrieval and displaying of the guidance TAFs in the frame of a given scenario.

Through the feedback of air users (pilots), the visualizer has been enriched with an alternate view for the data. Indeed, it appears that pilots need are much more focused on direct usability of the information (see Figure 14), whereas scientific/meteorological need is more focused on exhaustive provision of the information (see Figure 15).

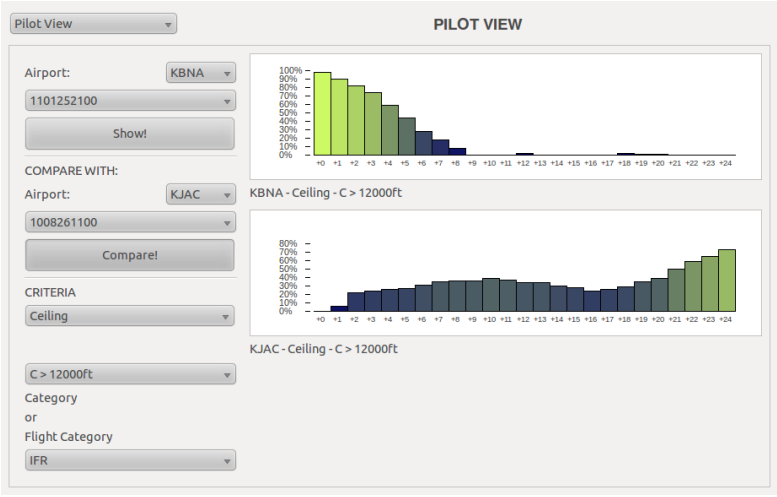

Figure 14 – Guidance TAF visualizer - Pilot View

Figure 15 – Guidance TAF visualizer - Scientific View

The Guidance TAF visualizer has been developed in C++. The graphical interface has been realized with Qt (set of C++ libraries for HMIs). The client is able to show any NAWX guidance TAF file in graphical.

The main challenges faced during the development of the TAF visualizer were:

- Getting access to the data through KVP queries from the WFS.
- Finding a proper way to display data to meet both meteorologist and pilot expectations. This has eventually been done by a segregation of the views

During the OWS-8 testbed, ATMOSPHERE has achieved the development of a Guidance TAF visualizer that encompasses both the pilot and the scientific perspective with regard to portrayal and display of Guidance Forecasts. The visualizer is now equipped with a comparison function, allowing to compare different TAF / airports with regard to a specific parameter. This, according to the pilot, could allow better selection of:

- Best time for starting time / duration / ending time of a flight.
- Best destination / alternate airport.

7.8 AIXM 5.1 Validation Tools

7.8.1 Lisasoft

DuckHawk is an open source testing framework that enables the development of automated reliability, load, performance, stress, error-handling and conformance tests. It can be used to automatically test any web service or application.

DuckHawk is written in Java and built upon JUnit 3. Using JUnit test runner functionality allows the framework to integrate with most commonly used build systems such as Maven and Ant and be used for continuous integration testing during product development. The tests themselves are also written in Java and use a design similar to

JUnit 3 tests. DuckHawk allows configuring the test parameters through simple configuration files, increasing the flexibility of the developed test systems.

DuckHawk can perform various types of testing (e.g., performance, reliability, etc.), but also provides modules for conformance testing (XML schema validation). Tests using the conformance testing module send requests to the target server and validate the returned XML response documents against an XML Schema.

DuckHawk collects all test results and passes them on to test listeners, which can perform processing, analysis, and generate reports. At the moment two main implementations of test listeners exist; an XML-report generator and a human readable HTML-report generator.

7.8.1.1 Schematron Rules

Included in the DuckHawk WFSValidator module are several Schematron files containing generic rules for validating the responses of the target servers. Some of these rules are part of OGC standards and some are written to validate general parts of an XML document (e.g., validation of dictionaries).

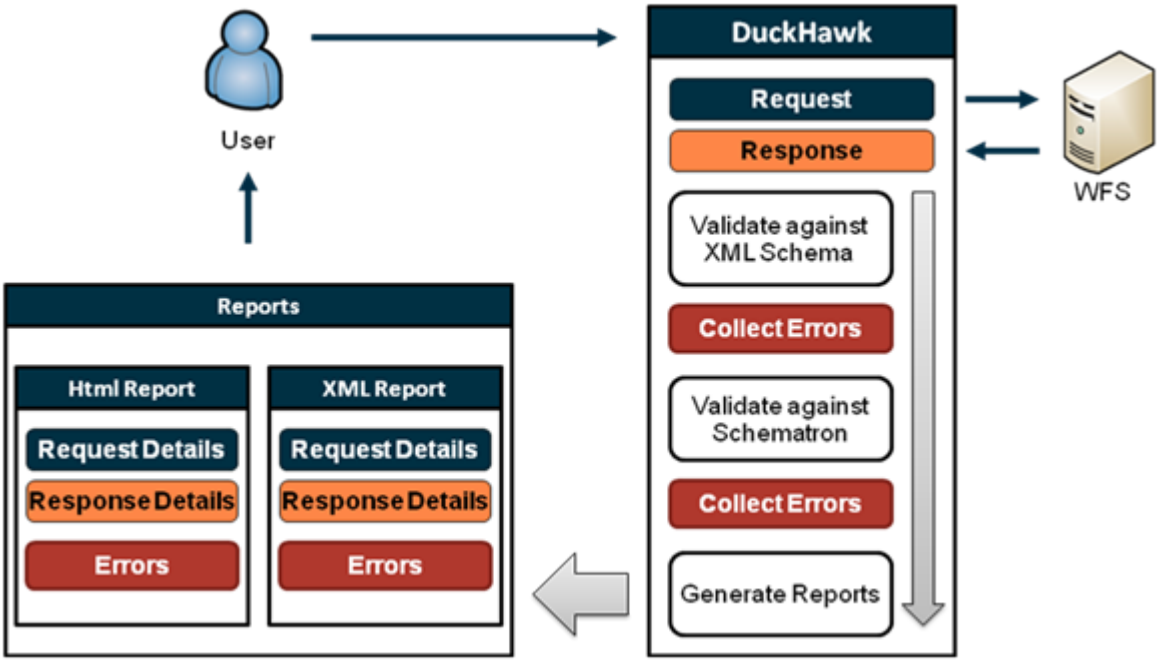

Figure 16 – DuckHawk Testing Framework – Overview

7.8.1.2 Use in OWS-8

For the OWS-8 testbed, the WFS Validator was preconfigured to perform validation of Digital NOTAM Events. To achieve this, the validator comes bundled with the Digital NOTAM schematron file developed during the testbed, as well as the latest XML schema files.

A new feature added to the validation tool in this testbed was to test local files as well as responses from a WFS. Included in the bundle are sample Digital NOTAM events developed to test the schematron file.

Currently the validation tool needs to be manually invoked, requiring user intervention. A proposal to encapsulate the validation tool functionality in a WPS service has been proposed in the Digital NOTAM Event Specification Engineering Report (see OGC 11-092). This would allow for automatic invocation of the validation tool, allowing for greater automation and interoperability.

7.9 AIXM Performance Assessment Tools

7.9.1 AtoS

The EXI-TTFMS platform, developed by W3C to compare the compression performance of EXI (Efficient XML Interchange) against other compression algorithms was the starting point of the AIXM benchmarking platform developed by AtoS. Analysis scripts were used to qualify data and for display purposes.

7.9.1.1 The EXI-TTFMS framework

7.9.1.1.1 Audience

This framework, developed upon Japex (micro-benchmarking platform developed by Oracle), is a key tool for a developer willing to:

- Appreciate the level of compaction reachable for an XML file through various existing and open source compression algorithms (Deflate, Fast Info Set, CWXML, EXI) using several options. The memory and CPU consumption is also communicated, given your specific hardware and the operating system.
- Develop a new AIXM compression algorithm, based on raw data or upon a SAX API, and check how it performs compared to existing ones.
- Modify an existing algorithm or tune specific parameters.
- Benefit from a large set of AIXM data files, from different sizes and nature, whose selection is the result of an analysis willing to identify the inner aspects of the AIXM structure.

7.9.1.1.2 Usage made simpler

We faced some difficulties to get a running framework, so we took care of providing a turn-key platform easy to use for a new “java” developer willing to add new AIXM files to the benchmark or a new algorithm. Components and candidates were updated to the latest versions publicly available (Japex, Fast Info Set, Xerces).

7.9.1.1.3 Improvements / replacements

The EXI commercial implementation of AgileDelta, was replaced by Siemens’s Exificient open-source EXI implementation

The framework has been modified to process raw Deflate instead of GZIP, with the possibility to use a pre-loaded dictionary and specify the compression level from 1 to 9.

CWXML (C library) is now supported through the JNI adapter of Japex.

Japex was modified to be able to generate graph including maximum memory consumption.

7.9.1.2 Test cases

Four families of AIXM files were identified and populated:

- A first family composed of small files (<10kB) : tree Digital Notice To AirMen (DNOTAMs)
- A second family composed of medium sized files (between 10kB and 1MB), each made from a single AIXM feature, bringing its own characteristic (for instance airspaces, geo borders, runways and taxiways elements contains much more coordinates than other features, routes have simple structure (only 28 different elements taking 65% of the file) compared to airspaces (60 different elements taking 30% of the file))
- A third family made of bigger files (>1MB), alternating both mixed features (like the whole Estonian database or the sum of all features from family 2) or single features to see how the performance of compression algorithms evolve along with volume
- A fourth family made of technical files, useful to check a specific aspect against all algorithms (influence of order, handling of autoclosing tags, dropping comments, formatting …).

7.9.1.3 Results

To sum up the results of the AIXM Compression ER (see OGC 11-097), we can conclude saying:

- EXI, using both schema knowledge and deflate post compression reduces a D-NOTAM to 13% of the size of the original file (with no more indenting and comments). This compression level provides DNOTAMs under 1KB, which allows their transmission using Very High Frequency (VHF) datalink (4 messages in Plain Old ACARS (POA), 1 single Aviation VHF Link Control (AVLC) default frame for Aeronautical Telecommunication Network (ATN) or ACARS over AVLC (AOA)).
- The building of the grammar is slow (seconds) and consumes memory (MBs), but is done only once at startup.
- The deflate post-compression implemented (in java) in Exificient is slow and can be improved.
- When the file size grows, EXI performance decays compared to other algorithms. Depending on the structure of the file (complexity, presence of free text, coordinates) it performs better than deflate only in a 20-50% range. This poor

performance in the long run, is due to AIXM Schema who offers too much free text and to the handling of floating points number by EXI (once converted in IEE 754 binary format, float and double are less likely to be compressed again by deflate in a second pass).

- Fast Info Set (FI) with deflate post-compression is the best value when we look closely at the ratio compression performance / CPU consumption. FI allows a faster SAX parsing. Its usage for WFS client/server interactions over a high speed ground network could be suitable.
- Surprising good performance (30% additional compression for DNOTAMs) was obtained by simply adding a dictionary based on AIXM XSD files to deflate. This could be a cheap way to get a decent compression performance using the well known Zlib compression layer.

7.9.1.4 Future work

Some aspects of WFS serialization are studied in the AIXM Performance Assessment ER, and some recommendation can be made to improve the action of compression:

- Sort output by feature when multiple features are requested
- Try to reduce the size of IDs
- Try to limit coordinates to float, and use double only when very necessary
- Remove BBOX from features
- Remove formatting, comments, …

To get further, maybe a subset of AIXM (profile) could be used to reduce EXI grammar and boost performance of DNOTAMs compression. However a perfect GML compression library would have to cope with coordinates and treat them apart (considering the dimension and order). A BZIP2 MTF kind of algorithm could be used with a differential encoding (like in FLAC) to get a real compression on coordinates without loss of precision.

7.10 Access Control System

7.10.1 TUM

The Policy Enforcement Point (PEP) – see section 8.3 for further details – is implemented as an Apache 2 Web Server configured as a Reverse Proxy. As such, it intercepts HTTP requests for a given URI (e.g. /service/WFS) and forwards the request to the appropriate Apache 2 Module.

The Context Handler is implemented as an Apache 2 Module which is loaded at Apache startup and executed if the Apache intercepts a WFS request on a given URI. In correspondence with the XACML information flow, the Context Handler creates the XACML Authorization Decision Request which is sent to the GeoPDP. The Context Handler is implemented according to the requirement classes &xop;/RC/1.2, &xop;/RC/1.3(&WFS:2.0;), &xop;/RC/1.4(&WFS:2.0;), &xop;/RC/1.9(&WFS:2.0;), &xop;/RC/1.11(&WFS:2.0;) defined in the XACML v2.0 OGC Web Service profile. For OWS-8, the Context Handler and the PIP are an instance for AIXM and as such

understands to resolve “Missing Attributes” for “aixm:controllingAgency” and “aixm:usingAgency”.

The GeoPDP is a Web Service that returns XACML Authorization Decision(s) upon an XACML Authorization Decision Request. The GeoPDP involved in OWS-8 is a GeoXACML v1.0 BASIC implementation including extensions A+B.

Next to the access control system components two demo clients have been implemented that show various access restrictions for the Snowflake and Comsoft Authoritative Data Store (also see the OWS-8 Aviation Thread - Authoritative AIXM Data Source Engineering Report - OGC 11-086).

8 Access Control System within the OWS-8 Aviation Architecture

8.1 Service-oriented Security Architecture

Separating security aspects as much as possible from the implementation of OGC Web Services allows securing existing OWS instances without security related code changes. This separation of concerns further enables leveraging available IT-security concepts and implementations.

When externalizing security functionalities it is advantageous to provide the security capabilities through separate security services (e.g. authentication, authorization and audit services). Security services can be flexibly combined and can be used in different configurations for several geo-processing services[3]. Each of these security services can itself be composed of further services.

Advantages of a modular security architecture approach are e.g.:

- Splitting the security solution into separated functional components reduces the associated development and maintenance complexity.
- The solution is fully scalable and easy to upgrade. New security services can be easily inserted and existing services can be upgraded without affecting the others.

Because of the mentioned advantages we use a service-oriented security architecture.

8.2 Initiation of the Access Control Process

During the design and development of an Access Control System (ACS) for an OWS based architecture one needs to address the question where to initiate the access control process in the overall system architecture. Figure 17 shows components (see ACS boxes) in which the access control process could be initiated.

[3] for further details, see Service Oriented Security Architecture applied to Spatial Data Infrastructures. Cristian OPINCARU, Munich 2008.

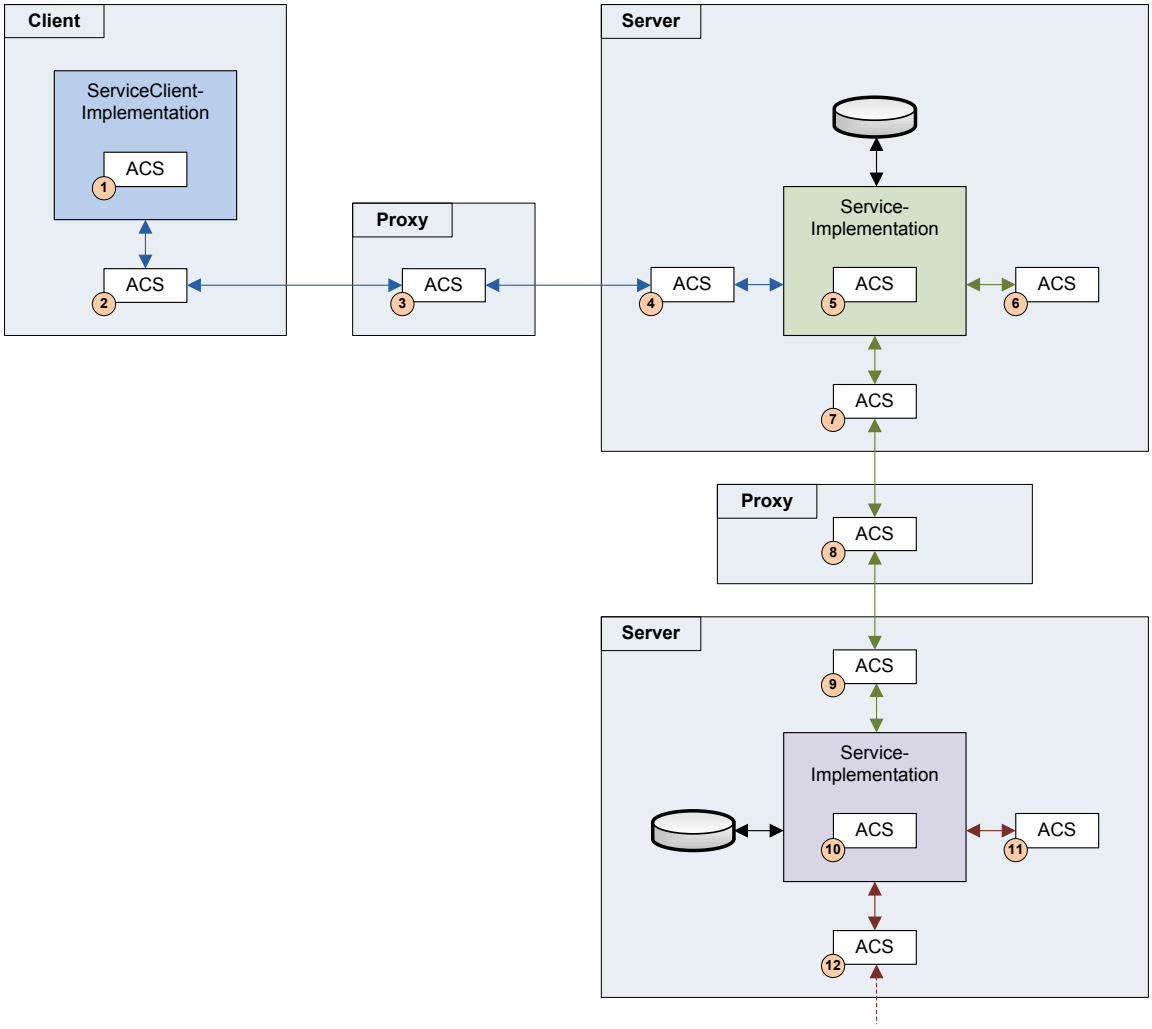

Figure 17 – Candidate components for the initialization of the access control process

No assumptions on the client side software configuration

In OWS based architectures one cannot assume that subjects interact with OWS instances through client programs with specific built-in security functionalities. It can e.g. be the case that subjects interact with services through ordinary web browsers. The consequence of this situation is that the access control process cannot be initiated and enforced in components labeled 1 and 2.

Access rights cannot be controlled "behind" services

Enforcing access rights in the components 6 to 12 implies that the access control process operates on the sub-requests and/or the corresponding responses. This is problematic in cases where the required authorization semantics can only be enforced based on the messages exchanged between the interacting subject and the service (e.g. GetCapabilities or WPS Execute operation requests). Next to this problem the post-service access control

approach is not realizable if the components 8 to 12 belong to other, independent administrative domains.

Independency of enforceable rights of the used service implementations

OWS implementations used in SOAs are usually proprietary, from different vendors and have no or variably powerful built-in access control capabilities. It cannot be assumed that all service implementations provide sufficiently expressive access control functionalities. Hence the access control process cannot be realized in the components 5 and 10.

The requirements listed above clearly reduce the number of suitable components and imply that the access control process can only be enforced in the components 3 or 4. This implies that an appropriate rights model must support the definition of rights that (next to others) refer to the intercepted messages. Whether component 3 (i.e. a dedicated proxy server) or component 4 (i.e. a server-side proxy component) is more suitable is dependent on the characteristics and requirements of the given use case. Component 4 could e.g. be in favor as it allows local calls of the access control system and thus implies certain performance advantages. In contrast, the initialization of the access control process in component 3 can result in scalability and availability advantages.

8.3 Architecture of XACML based Access Control Systems

Figure 18 shows the architecture of the proposed rule- and role-based access control system and gives a rough impression of the internal information flow. The access control system serves as a proxy component that intercepts messages exchanged between subjects and WFS instances.

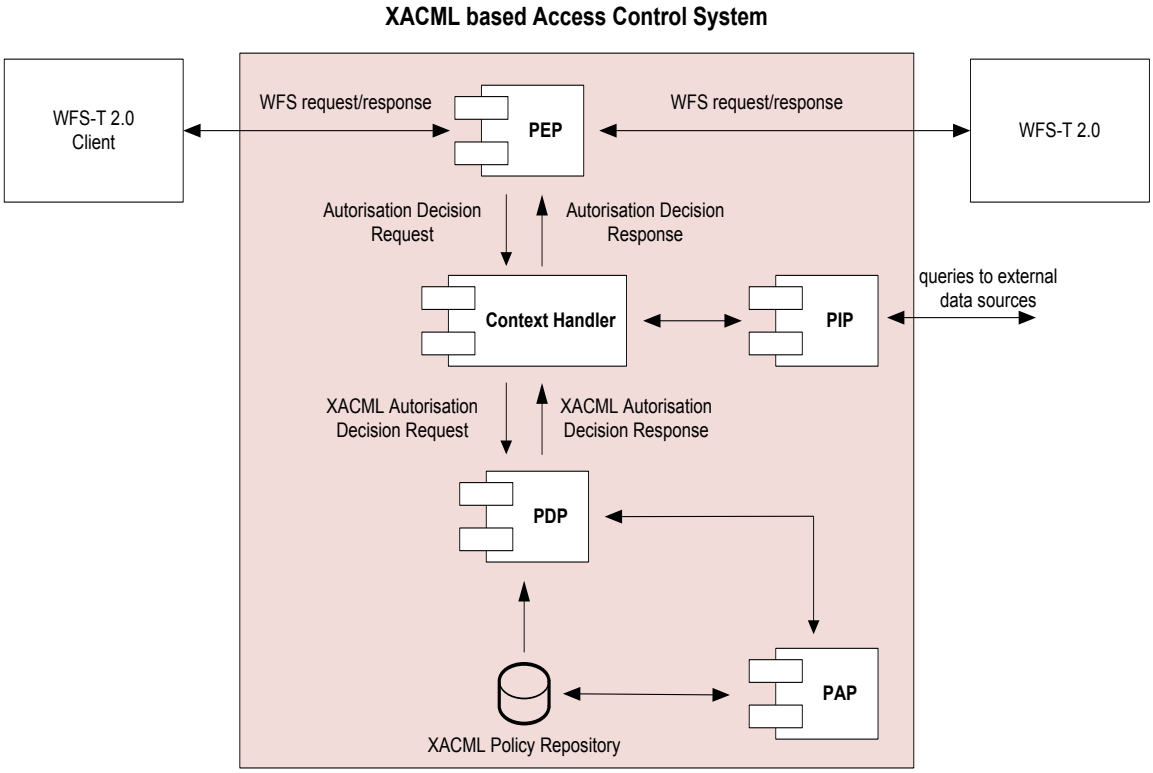

Figure 18 – Architecture of an XACML based Access Control System

The **Policy Enforcement Point (PEP)** is the entry point in the access control process. Based on an intercepted WFS request or response, the PEP generates an authorization decision request in an implementation specific (or already XACML compliant) format and sends it to the **Context Handler**.

When receiving an authorization decision request from a PEP the Context Handler generates an XACML authorization decision request (XACML ADR) based on the information already included in the received request and - if required - based on additional information that can be queried from external information sources through the **Policy Information Point (PIP)**.

The Context Handler forwards the generated XACML ADR to the **Policy Decision Point (PDP)**. The PDP evaluates the incoming ADR by searching for applicable rules defined in the currently loaded XACML policy. The effects of all rules that evaluate to ‘true’ or ‘indeterminate’ under the given ADR are combined and an XACML authorization decision response is returned to the Context Handler.

The Context Handler interprets the result returned by the PDP and acts correspondingly. In the end the Context Handler translates the XACML encoded authorization decision response back into the application specific authorization decision request/response language (if this is not XACML) and will then forward the response to the PEP. The PEP will in turn act according to the result of the access control process.

The **Policy Administration Point (PAP)** is the component that allows policy administrators to retrieve, insert, update, delete, test and analyze XACML encoded access rights. Additionally the PAP can be used by the PDPs to query relevant parts of XACML policies.

It is important to highlight that the presented architecture of rule-based access control systems is very flexible. Each of the introduced components can be replicated and distributed as required. In addition, certain components can be aggregated into one component. For example, one can implement a PEP that consolidates the PEP and Context Handler functionality.

More detailed information on the information flow within XACML based access control systems and on the implementation of the various components of the access control system can be found in the OWS-8 Aviation Thread - Authoritative AIXM Data Source Engineering Report (OGC 11-086).

9 Aviation Event Architecture

This section describes the architecture for the dynamic processing of digital NOTAM events used in OWS-8. This includes the description of the used event encodings as well as the involved components. An overview on achieved innovations, namely the dynamic spatial filtering of DNOTAMs and the enrichment of thin DNOTAMs with unchanged information, will be presented in this section. In addition, the issues and drawbacks which have been observed during OWS-8 are summarized.

9.1 Encoding of Aviation Events

The communication between the involved components makes use of two concepts. Temporal changes to the AIXM features contained in the WFS data stores are encoded and published using the dedicated Event extension of AIXM 5.1 as specified by the Digital NOTAM Event Specification 1.0. As the Event Service is based on the OASIS Web Services Notification (WS-N) standards family the components involved in the Event Architecture use these standards as the transportation protocol. The latter is described in section 9.2 in more detail.

9.1.1 Digital NOTAM

The Event Architecture of OWS-8 Aviation follows the guidance of the Digital NOTAM Event Specification 1.0 on encoding NOTAM events. The AIXMBasicMessage element is used as a container for the Event feature itself and the affected AIXM features. Figure 19 shows an Event affecting an Airspace contained in such a message container. The Event defines the time at which the happening is valid. The Airspace element contains one or more timeslices with the changed information as well as the link to the Event using the 'theEvent' element defined by the DNES.

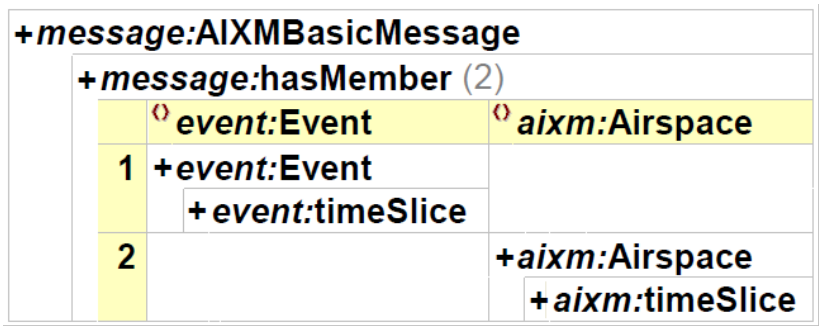

Figure 19 – AIXMBasicMessage transporting a Digital NOTAM

9.1.1.1 Temporality and Uniqueness

The architecture is capable of handling data which is produced in due consideration to the specifications of the AIXM 5.1 Temporality Model by defining the interpretations of Events as pairs of Baseline/Permdelta or Tempdelta timeslices. As opposed to OWS-7, in OWS-8 only the data which is changed is provided by the Events (see also section 9.5.1).

Similar to other AIXM features an Event uses the gml:identifier element as a unique identifier. Identifying an Event is necessary for cases when an Event has to be canceled

or updated due to informational errors. This was one of the encoding recommendations of the OWS-7 Aviation Architecture Engineering Report (OGC 10-079) and has also been documented as a recommendation in the OWS-8 Report on Digital NOTAM Event Specification (OGC 11-092).

9.1.1.2 Spatial Extent

Similar to the previous testbeds the geometries of AIXM features are – besides their actual spatial extent – encoded using the gml:boundedBy element. As most of AIXM features have a specific or potentially no geometry at all, the workaround as established in OWS-6 and OWS-7 has been used in this testbed as well. Here, bounding boxes have been created by the service provider explicitly and included in the DNOTAM. The bounding box was then used for spatial filtering. The WFS data stores are responsible for computing the relevant geometry for every feature to enable spatial filtering of digital NOTAMs using the Event Service. There are several issues (e.g. collections of or multiple geometries per feature; as discussed in OGC 10-079) which have not been addressed during this testbed. For most cases spatial filtering is performed using the computed bounding box (e.g. of an Airspace) or special geometry forms such as DirectPositions (e.g. aixm:ElevatedPoint for a Navaid) of an AIXM feature.

9.2 Web Service Notification and SOAP

For transmission of Events the WS-N communication patterns are used. This results from the fact that the Event Service – acting as an intermediary component for data filtering (see section 7.4) – is based on the OASIS Web Services Notification (WS-N) standards family. SOAP version 1.2 was applied as the protocol binding for the involved WS-N components. Within this testbed version SOAP 1.2 has been used in combination with WSDL 1.1 descriptions of the Event Services to enable SOAP bootstrapping (see section 10.2.2). WS-N defines a set of roles for components of a distributed service architecture and message protocol. In particular, these roles are Publishers, NotificationProducers, NotificationConsumers and NotificationBrokers which are briefly described in the following:

- A Publisher is responsible for formatting a Notification and disseminating it to a NotificationProducer
- A NotificationConsumer must provide an interface for receiving NotificationMessages
- A NotificationProducer must provide an interface for subscribing to a subset of messages and must be capable to deliver these subsets of NotificationMessages to the subscribing NotificationConsumer
- A NotificationBroker combines the tasks of a NotificationProducer and a NotificationConsumer and acts as an intermediary distributor.

The message protocol defines a NotificationMessage carrying the contents of an event (such as a DNOTAM) in a dedicated message element. Examples of request/response communication as well as push-based communication are presented in the following section.

A NotificationMessage can also define a topic for its contents. The Event Architecture makes use of these topics for grouping similar events as recommended in the OWS-7 Aviation Architecture Engineering Report (OGC 10-079). The following topics (also called *channels*) were used in the testbed:

- dNOTAM-Events
- AircraftPositionUpdates.

Thus, a NotificationConsumer can easily subscribe for a subset of similar events disseminated on the same channel.

9.3 Eventing Components and Dataflow

This section describes the components involved in the event architecture and their roles in detail. In general, the architecture can be separated into three tiers as done in chapter 5. The Event Architecture only uses the Event Services for processing and disseminating DNOTAM data. Their role can be summarized as an information broker within the general OWS-8 Aviation Architecture.

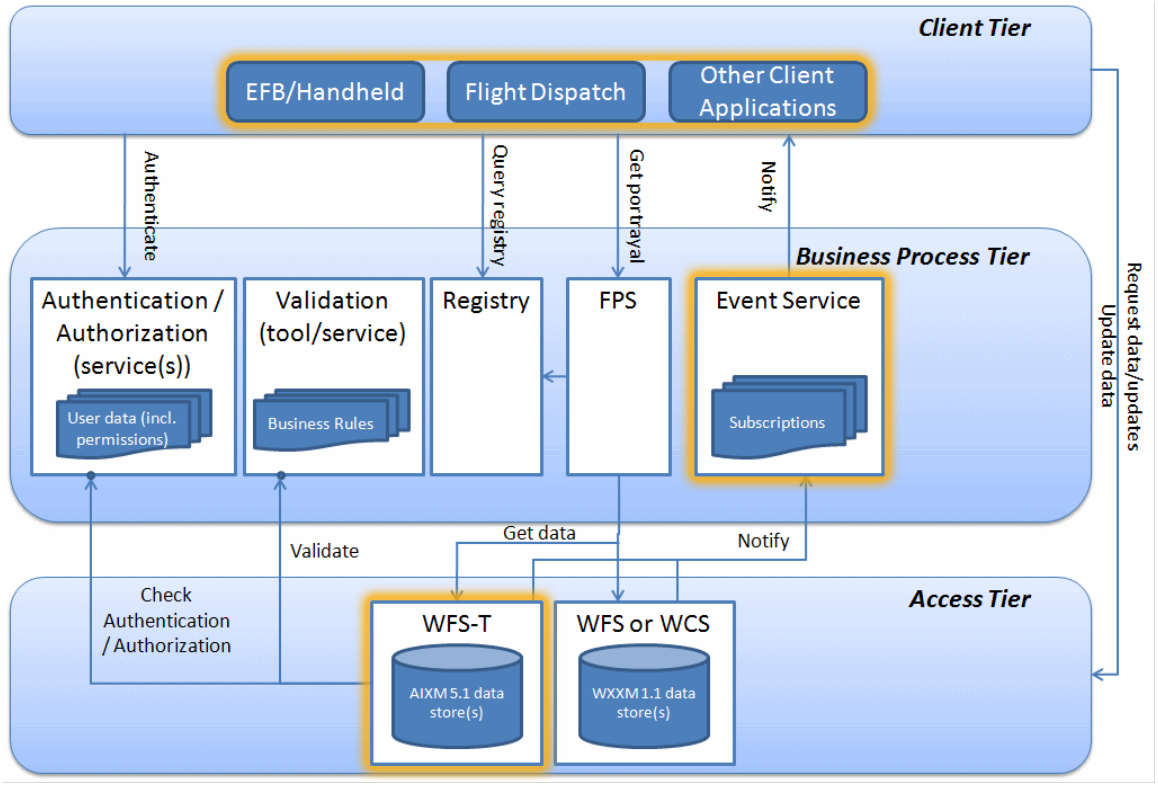

Figure 20 – Components involved in OWS-8 Eventing

Figure 20 depicts the components used in the eventing parts of OWS-8 (based on the corresponding figure in chapter 5. Figure 21 gives a high-level overview on the interaction patterns between the components.

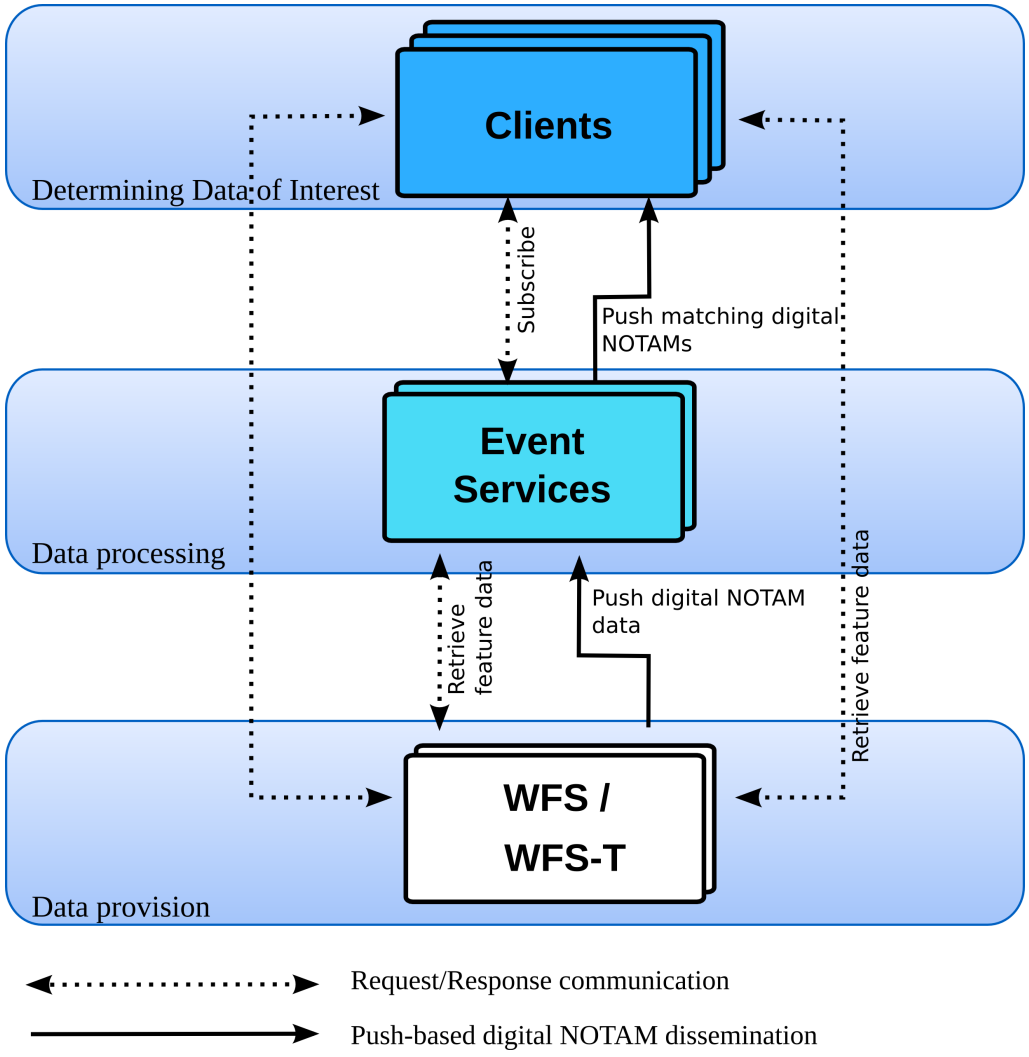

Figure 21 – Data flow between components of the Event Architecture

9.3.1 Determining Data of Interest

The client components are responsible for defining the digital NOTAMs that are of interest. They act as WS-N NotificationConsumers and thus must provide a valid NotificationConsumer endpoint to which an Event Service can deliver digital NOTAMs. For the definition of reasonable subsets of all digital NOTAMs the clients make a subscription using the following filters:

- XPath 1.0
- Topic or channel filter
- OpenGIS Filter Encoding Specification (FES) 2.0
- Event Pattern Markup Language (see section 9.4).

Listing 1 – XPath subscription

```
<wsnt:Subscribe  xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <wsnt:ConsumerReference>
    <wsa:Address>{client-url}</wsa:Address>
  </wsnt:ConsumerReference>
  <wsnt:Filter>
    <wsnt:TopicExpression Dialect="http://docs.oasis-open.org/wsn/t-
1/TopicExpression/Simple">
      dNOTAM-Events
    </wsnt:TopicExpression>
    <wsnt:MessageContent Dialect="http://www.w3.org/TR/1999/REC-xpath-19991116">
      //aixm:designator="LAX"
    </wsnt:MessageContent>
  </wsnt:Filter>
</wsnt:Subscribe>
```

Listing 1 shows a subscription to select all digital NOTAMs concerning features with the designator set to 'LAX' using an XPath expression. The subscription is additionally restricted to only select messages posted on the 'dNOTAM-Events' channel. This example shows that multiple filters are allowed in one subscription. Multiple filters are combined using a logical AND.

Listing 2 – FES 2.0 subscription

```
<wsnt:Subscribe xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <wsnt:ConsumerReference>
    <wsa:Address>{client-url}</wsa:Address>
  </wsnt:ConsumerReference>
  <wsnt:Filter>
    <wsnt:MessageContent Dialect="http://www.opengis.net/ses/filter/level2">
      <fes:Filter xmlns:fes="http://www.opengis.net/fes/2.0">
        <fes:DWithin>
          <fes:ValueReference>{path_to_dnotam_geometry}</fes:ValueReference>
            <gml:LineString gml:id="flight_route_OWS-8_A"
srsName="urn:ogc:def:crs:OGC:1.3:CRS84" xmlns:gml="http://www.opengis.net/gml/3.2">
              <gml:coordinates decimal="." cs="," ts=" ">
                -105.873,45.559 -105.963,45.543
              </gml:coordinates>
            </gml:LineString>
          <fes:Distance uom="[nmi_i]">500</fes:Distance>
        </fes:DWithin>
      </fes:Filter>
    </wsnt:MessageContent>
  </wsnt:Filter>
  <wsnt:InitialTerminationTime>PT5H</wsnt:InitialTerminationTime>
</wsnt:Subscribe>
```

The subscription shown in Listing 2 makes use of a FES 2.0 spatial filter. All digital NOTAMs with an included or related geometry within a 500 nautical mile buffer around the specified flight route are selected by this filter. Additionally, this subscription defines a maximum lifetime of 5 hours. This allows the clients to automatically schedule the termination of each subscription (e.g. terminate the subscription at the end of the estimated flight time). Time durations (as xsd:duration) as well as date times (as xsd:dateTime, following chapter 5.4 of ISO 8601) are valid arguments.

Using EML for defining a subset of Events is described in section 9.4.

9.3.2 Data Provision

The Event Architecture is based on the provision of digital NOTAMs as described in the previous sections. The (transactional) WFS instances provide all digital NOTAM events

and act as WS-N Publishers. Such events are pushed to the Event Services when they become available at the WFSs. The WFS servers also act as a static repository for retrieval of AIXM features and snapshots as described in section 9.5.1.

Listing 3 – Example digital NOTAM Notification

```
<wsnt:Notify xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">
  <wsnt:NotificationMessage>
    <wsnt:Message>
      <msg:AIXMBasicMessage gml:id="gmlID14726"
      xmlns:msg="http://www.aixm.aero/schema/5.1/message"
xmlns:gml="http://www.opengis.net/gml/3.2" xmlns:aixm="http://www.aixm.aero/schema/5.1"
xmlns:dnotam="http://www.aixm.aero/schema/5.1/event">
        <msg:hasMember>
          <dnotam:Event>
            <gml:identifier codeSpace="urn:uuid:">BCB4F904-9AEA-4F2E-97AE-
2D4A4344BD6C</gml:identifier>
            <dnotam:timeSlice>
            ...
            </dnotam:timeSlice>
          </dnotam:Event>
        </msg:hasMember>
        <msg:hasMember>
          <aixm:Navaid gml:id="gmlID14730">
            <gml:identifier codeSpace="urn:uuid:">BCB4F904-9AEA-4F2E-97AE-
2D4A4344BD6D</gml:identifier>
            <aixm:timeSlice>
              <aixm:NavaidTimeSlice gml:id="gmlID14731">
              ...
              <aixm:interpretation>TEMPDELTA</aixm:interpretation>
              <aixm:sequenceNumber>1</aixm:sequenceNumber>
              <aixm:correctionNumber>5000</aixm:correctionNumber>
              ...
              <aixm:availability>
              <aixm:NavaidOperationalStatus gml:id="gmlID14734">
              <aixm:operationalStatus>UNSERVICEABLE</aixm:operationalStatus>
              </aixm:NavaidOperationalStatus>
              </aixm:availability>
                <aixm:extension>
                  <dnotam:NavaidExtension gml:id="gmlID14735">
                    <dnotam:theEvent xlink:href="urn:uuid:BCB4F904-9AEA-4F2E-97AE-
2D4A4344BD6C"/>
                  </dnotam:NavaidExtension>
                </aixm:extension>
              </aixm:NavaidTimeSlice>
            </aixm:timeSlice>
          </aixm:Navaid>
        </msg:hasMember>
      </msg:AIXMBasicMessage>
    </wsnt:Message>
  </wsnt:NotificationMessage>
</wsnt:Notify>
```

Listing 3 populates a “Navaid unservicable” DNOTAM using a Tempdelta. It uses the NavaidExtension to link to the Event feature contained in the same XML document.

9.3.3 Data Processing

The actual processing is done by the Event Services. All incoming digital NOTAMs are checked against subscriptions (more specifically: the filters they define) created by the clients. If a filter matches for a certain subscription the originally received DNOTAM is pushed to the client without modifying it. This must also be the case if the Event Service applied enrichment of thin events (see section 9.5.1).

9.4 Dynamic Filtering

The OWS-7 Aviation Architecture Engineering Report (OGC 10-079, Section 8.4.5) introduced the concept of a dynamic filter for DNOTAM events. During OWS-8 the

details for such a concept have been developed and implemented prototypically. Figure 22 illustrates the idea of this concept.

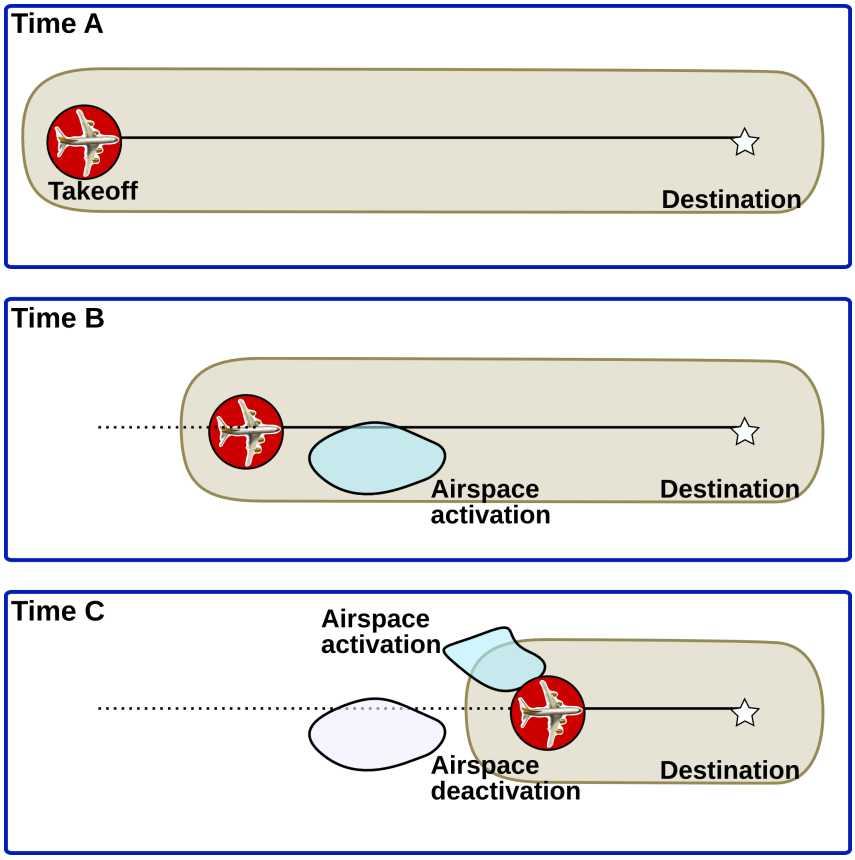

Figure 22 – Dynamic buffer of a flight route

As illustrated in this example a client receives a notification about the activation of an Airspace (Time B). Later (Time C) another Airspace is activated and the formerly activated Airspace gets inactive again. The clients do not receive notifications for these events as the buffer (used as filter geometry) is updated dynamically using the position updates received by the aircraft.

By updating a dynamic subscription using position data a client does not have to manage multiple subscriptions for each route portion and no further request/response communication to update the filter geometry is needed. In addition, another advantage of such a subscription is that no unnecessary information is delivered to the clients. Hence bandwidth is not wasted and the pilot or dispatcher is not distracted by irrelevant DNOTAMs.

9.4.1 Creation of a Dynamic Spatial Filters

The support for dynamic subscriptions in the Event Service in general can be realized using the Event Pattern Markup Language (EML). EML is currently a discussion paper at OGC (OGC 08-132) and several improvements happened to it since its submission. The concept is based on Complex Event Processing (CEP) to define patterns of Events.

In this testbed the dynamic spatial filter is realized using an extension of the original EML. A generic restrictive view has been introduced in the EML model. This generic view enables the dynamic adjustment of a pattern using a defined set of variable parameters. Listing 4 illustrates the generic view used in this testbed.

Listing 4 – Generic restrictive view

```
<wsnt:Filter>
  <wsnt:MessageContent Dialect="http://www.opengis.net/ses/filter/level3">
    <eml:EML xmlns:fes="http://www.opengis.net/fes/2.0"
xmlns:eml="http://www.opengis.net/eml/0.0.2"
xmlns:swe="http://www.opengis.net/swe/1.0.1">
     <eml:SimplePatterns>
     <eml:SimplePattern patternID="select_aircraft_updates">
      ...
      <eml:PropertyRestrictions>
        <!-- use the CallSign to determine the needed position updates -->
        <eml:PropertyRestriction>
        <eml:name>dynamic_filter_stream/callSign</eml:name>
        <eml:value>OWS-8_A</eml:value>
        </eml:PropertyRestriction>
      </eml:PropertyRestrictions>
      ...
      </eml:SimplePattern>
      ...
    </eml:SimplePatterns>
    <eml:ComplexPatterns>
     <eml:ComplexPattern patternID="dynamic_buffer_using_aircraft_positions">
      <!-- select function to select the current and former aircraft position-->
      <eml:SelectFunctions>
        <eml:SelectFunction newEventName="geometry_in_dynamic_buffer"
outputName="geometry_in_dynamic_buffer_stream">
        <eml:UserDefinedSelectFunction name="SelectGeometryInDynamicBuffer">
          <eml:FunctionParameters>
          <!-- set the former_position = current_position -->
          <eml:FunctionParameter>
            <eml:UserParameterName>former_position</eml:UserParameterName>
            <eml:UserParameterValue>current_position</eml:UserParameterValue>
          </eml:FunctionParameter>
          <!-- set the current_position = new position input -->
          <eml:FunctionParameter>
            <eml:UserParameterName>current_position</eml:UserParameterName>

<eml:UserParameterValue>aircraftPositionStream/gml:pos</eml:UserParameterValue>
          </eml:FunctionParameter>
          </eml:FunctionParameters>
        </eml:UserDefinedSelectFunction>
        </eml:SelectFunction>
       </eml:SelectFunctions>
      <!-- the Gerenic view -->
      <eml:View>
        <eml:GenericView>
        <eml:ParameterDefinitions>
          <!-- which parameters affect the view? -->
          <eml:ParameterDefinition>
<eml:ParameterIdentifier>aircraftPositionStream/aircraftPosition</eml:ParameterI
dentifier>
          </eml:ParameterDefinition>
          <eml:ParameterDefinition>
          <eml:ParameterIdentifier>allData/geometry</eml:ParameterIdentifier>
          </eml:ParameterDefinition>
        </eml:ParameterDefinitions>
        <eml:InsertCriteria>
          <eml:InsertCriterion>
          <fes:Filter>
            <fes:DWithin>
            <fes:ValueReference>//schema-
element(gml:AbstractGeometricPrimitive)</fes:ValueReference>
            <!-- the static flight route -->
            <gml:Curve gml:id="ls1" srsName="urn:ogc:def:crs:OGC:1.3:CRS84">
              <gml:segments>
                <gml:GeodesicString>
                  <gml:posList>24.8242444289068 59.41329527536156 -87.90381968189912
41.97626011616167 -157.91798998500212 21.322043902734308</gml:posList>
                </gml:GeodesicString>
              </gml:segments>
```



```
            </gml:Curve>
            <fes:Distance uom="[nmi_i]">500</fes:Distance>
            </fes:DWithin>
          </fes:Filter>
          </eml:InsertCriterion>
        </eml:InsertCriteria>
        ...
        </eml:GenericView>
      </eml:View>
      <eml:OR/>
      <eml:FirstPattern>
      <eml:PatternReference>select_aircraft_updates</eml:PatternReference>
      <eml:SelectFunctionNumber>0</eml:SelectFunctionNumber>
      </eml:FirstPattern>
      <eml:SecondPattern>
      <eml:PatternReference>select_all_data</eml:PatternReference>
      <eml:SelectFunctionNumber>0</eml:SelectFunctionNumber>
      </eml:SecondPattern>
      </eml:ComplexPattern>
    </eml:ComplexPatterns>
    <eml:TimerPatterns/>
    <eml:RepetitivePatterns/>
    </eml:EML>
  </wsnt:MessageContent>
</wsnt:Filter>
```

The generic view defines the parameters which affect the state of the filter using the ParameterDefinitions element. Additionally, the Event Service should not process any incoming aircraft position updates but only those which affect the appropriate flight. To ensure this, the incoming position updates are restricted using the PropertyIsEqualTo filter to only match updates of aircrafts with a CallSign “OWS-8_A”.

A client interested in a dynamic spatial filter can use this EML markup as a filter in the subscription (see section 9.3.1). The subscription using EML builds the external interface and an Event Service implementation can determine that a subscription uses input parameters to update the state of its geometry (see following figure).

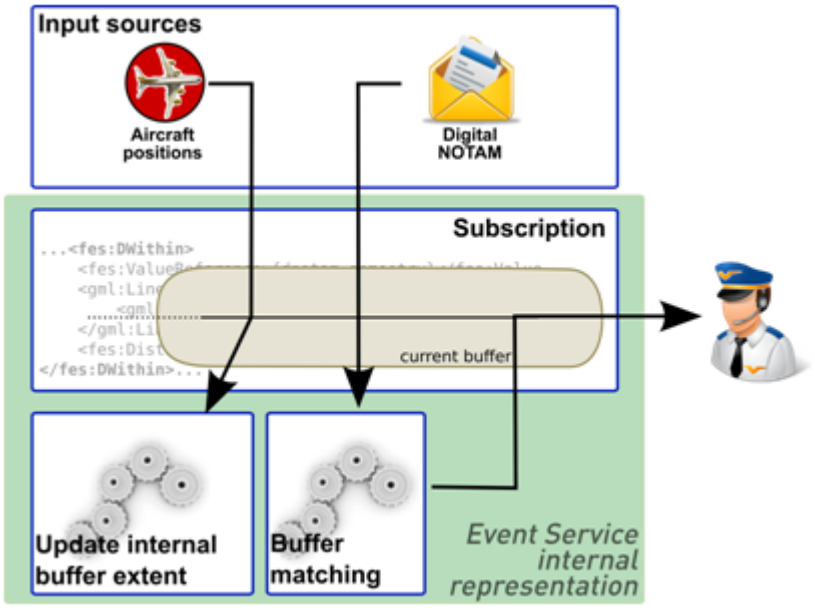

Figure 23 – State tracking of the dynamic spatial buffer

9.4.2 Provision of Aircraft Position Data

For demo purposes the OWS-8 clients simulate the broadcasting of the aircraft position using a dedicated encoding defined particularly for this purpose (see Listing 5). In a realistic scenario in the future the dynamic spatial filter could use the ADS-B[4] messages as input for aircraft position updates.

Listing 5 – Update for the position of an aircraft

```
<AircraftPositionUpdate xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <CallSign>OWS8 A</CallSign>
  <Position>
    <gml:pos srsName="urn:ogc:crs:epsg:4326">58.04 0.454</gml:pos>
  </Position>
</AircraftPositionUpdate>
```

This encoding covers all data items necessary for demoing the dynamic spatial filter. The call sign of an aircraft is defined by the CallSign element and the current position of the aircraft is encoded using the GML DirectPositionType “gml:pos”. There is an additional optional element to assign the time of creation. This time element can be used to ensure a certain level of temporal ordering of the incoming position updates. This can be helpful for the correct computation of the spatial filter (see section 9.4.3) as well as determining transportation delay and for logging purposes.

9.4.3 Processing of Position Data

In addition to the external interface an Event Service supporting dynamic spatial filters must be able to update the internal state, namely the spatial buffer area, of subscription filters. During OWS-8 several issues and open questions were identified that need to be addressed in future work:

- As probably the flight route is calculated in an approximated way inside an Event Service implementation (e.g. LineString) an appropriate way of “deactivating” route segments is needed. This could be achieved by calculating distances to route segment points and corresponding comparison to previously calculated distances. If the distance increases (the plane has passed the segments endpoint), the route segment could be deactivated. Additionally, an airplane never passes a flight route perfectly exact. Thus, position updates or the flight route should incorporate some sort of tolerance radius to avoid erroneous deactivation of a flight route portion.
- If an airplane swings off the planned route for some reason, how should the dynamic spatial filter react? Should the complete subscription be canceled? Or should it be paused until the airplane is back on the planned route?

9.5 Observed Issues and Drawbacks

During OWS-8 several issues occurred within the Event Architecture. This section describes these issues and illustrates possible solutions.

---

4 http://www.eurocontrol.int/cascade/public/subsite_homepage/homepage.html, http://www.faa.gov/nextgen/portfolio/trans_support_progs/adsb/

9.5.1 Dealing with the AIXM Temporality Model

In the OWS-7 testbed a workaround was applied to provide all data to the Event Service needed for applying filtering. In particular, the geometries of features were included in DNOTAMs (containing delta timeslices), although the geometries themselves did not change. This is not conform to the intention of the AIXM Temporality Model and treating events published to an Event Service this way should be avoided as emphasized by the Digital NOTAM Event Specification. Still, Event Service instances should be able to do filtering and complex processing on any property of a feature to support more complex filtering. To overcome this issue an approach has been discussed within this testbed. It is described in the following sections.

9.5.1.1 Event Service Conformance Classes

Conformance classes define the level of information needed by an Event Service to apply appropriate filters. Three conformance classes have been defined for the Event Service: basic, snapshot and full-info.

- **Basic** means that an Event Service can only filter on the data provided by (thin) DNOTAM events following the AIXM Temporality Model. In particular, only the changed data is available for filtering.
- **Full-info** means that an Event Service has enrichment capabilities. The Event Service is aware of an appropriate (WFS) data store and requests all needed data for an (AIXM) feature of a thin DNOTAM.
- **Snapshot** is a conformance class in between the former two. An event publisher should provide snapshots of features to the Event Service who can then support subscriptions that access all properties of an AIXM feature at a given time (though only at the time instant represented by that snapshot).

The communication patterns for the “Snapshot” conformance class do not follow the AIXM Temporality Model as event publishers would provide information which does not change. But still there are use cases (probably beyond AIXM) where such Snapshots can be used in an Event Architecture. Within OWS-8 the Event Services either implemented the Full-info class or stuck to the Basic class.

9.5.1.2 WFS Support for Dynamic Features

To support the Full-info conformance class a WFS data store must support the retrieval of all information of an AIXM feature (e.g. Snapshot). The support for snapshots has been discussed and implemented within this testbed. Details can be found in the *OWS-8 Aviation: Guidance for Retrieving AIXM 5.1 data via an OGC WFS 2.0* Engineering Report (OGC 11-073). Nevertheless, it has to be analyzed if a single snapshot is sufficient for such use cases. An event has a time period as its validTime. Hence, a single snapshot does not always represent every possible state of a feature for the time period of an event. An Event Service should therefore be able to request all states of the feature and based on these states decide if an Event matches certain filter criteria. A general solution is needed for such decisions (e.g. an Event matches if there exists one state of the feature for which the criteria match).

9.5.2 Interoperability between Event Services

Another issue which came up during the integration of both the IfGI and IDS Event Service into the client applications was the different way of handling subscription resources. Such resources were required to enable management of subscriptions on the client-side (e.g. pausing or removing a subscription).

Listing 6 – Subscribe response

```
<wsnt:SubscribeResponse xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <wsnt:SubscriptionReference>
    <wsa:Address>http://v-tml.uni-
muenster.de:8080/EventService/services/SubscriptionManagerContextPath</wsa:Address>
    <wsa:ReferenceParameters>
      <es-wsa:ResourceId xmlns:es-
wsa="http://www.opengeospatial.org/projects/initiatives/ows-8/es">
        Resource-7
      </es-wsa:ResourceId>
    </wsa:ReferenceParameters>
  </wsnt:SubscriptionReference>
  <wsnt:CurrentTime>2011-08-17T12:47:33+02:00</wsnt:CurrentTime>
  <wsnt:TerminationTime>2011-08-17T13:47:33+02:00</wsnt:TerminationTime>
</wsnt:SubscribeResponse>
```

Listing 6 depicts the way both Event Services designate the resource of the subscription. The WS-N standards family does explicitly define a way how to designate subscription resources by using service-specific parameters and because the Event Services is based on these standards using a proprietary element to identify resources is valid. During OWS-8 the management of subscription resources on the client-side led to some minor problems as both Event Services used different namespaces for the ResourceId element. This was probably due to first-time application of WS-A specific policies in an OWS service architecture. As a workaround both Event Services used the same namespace. However, such a restriction is not necessary and may complicate the integration of additional services into the Event Architecture as these would need to apply domain-specific policies on a domain-unspecific environment such as WS-A.

Two possible approaches have been discussed and can be applied in future testbeds. One approach would be to stay with the current design and delegate the responsibility of managing the different namespaces to the clients which could address the complexity issues with existing software libraries and components. The other solution would be URL-based resource management (e.g. one specific URL for managing each subscription). This would foster the interoperability among different Event Service implementations, but would also imply a modification of the underlying communication binding at the Event Service.

9.5.3 Additional Observations

The Digital NOTAM Event Specification states that “Missing a Digital NOTAM may be safety critical”. Regarding the Event Architecture there can be cases that a client who receives DNOTAMs using subscriptions at an Event Service misses a DNOTAM due to the filter of the subscription. For instance, an Event had been published to create an ad-hoc restricted airspace and the included spatial extent does not match the spatial filter criteria of the subscription. Due to a mistake in the definition of the spatial extent of this

Event an additional Event for correcting the geometry is being created. The corrected geometry now matches the filter criteria of the client's subscription but the client cannot resolve the reference to the former published ad-hoc airspace feature. Although this situation presumably is a rare case the issue should be discussed and a feasible solution needs to be developed within the Event Architecture as the issue can affect security.

10 Lessons Learned

10.1 AIXM / Temporality Model

10.1.1 Clarify Snapshot Definition

10.1.1.1 Problem Statement and Description

Section 2.5 of the Temporality Model defines a Snapshot as follows:

"*SNAPSHOT = A kind of Time Slice that describes the state of a feature at a time instant, as result of combining the actual BASELINE Time Slice effective at that time instant with all TEMPDELTA Time Slices that are effective at that time instant.*"

This definition does not mention Permdeltas. Section 2.6 of the Temporality Model states the following:

"*From a conceptual point of view, a PERMDELTA Time Slice occurs at the edge between any two consecutive BASELINE Time Slices and it contains values strictly for the changed properties. [...] Conceptually, there exists a direct dependence between PERMDELTA and BASELINE Time Slices. However, this does not mean that the BASELINE Time Slice needs to be effectively instantiated after each PERMDELTA. In an implementation, it is possible, for example, to "accumulate" PERMDELTA Time Slices. The instantiation of a new BASELINE might occur, for example, after each third PERMDELTA affecting a feature.*"

Apparently, the Temporality Model allows the static property values given in a Baseline to be superseded by changes introduced through Permdeltas, also during the validTime of the Baseline.

Consider the following figure.

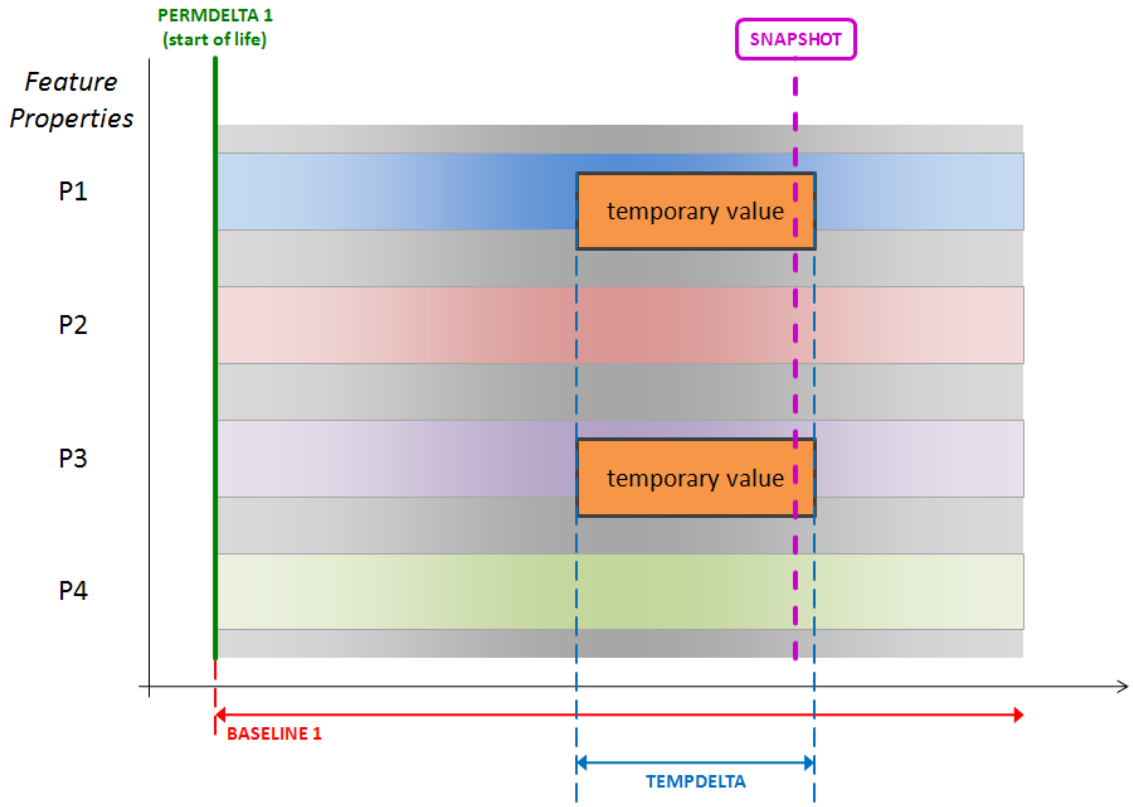

Figure 24 – Snapshot with Baseline and Tempdelta

The figure shows a Snapshot that takes into account property values as defined by both a Baseline and a Tempdelta timeslice. The current Snapshot definition of the Temporality Model covers this situation. Now consider the next figure.

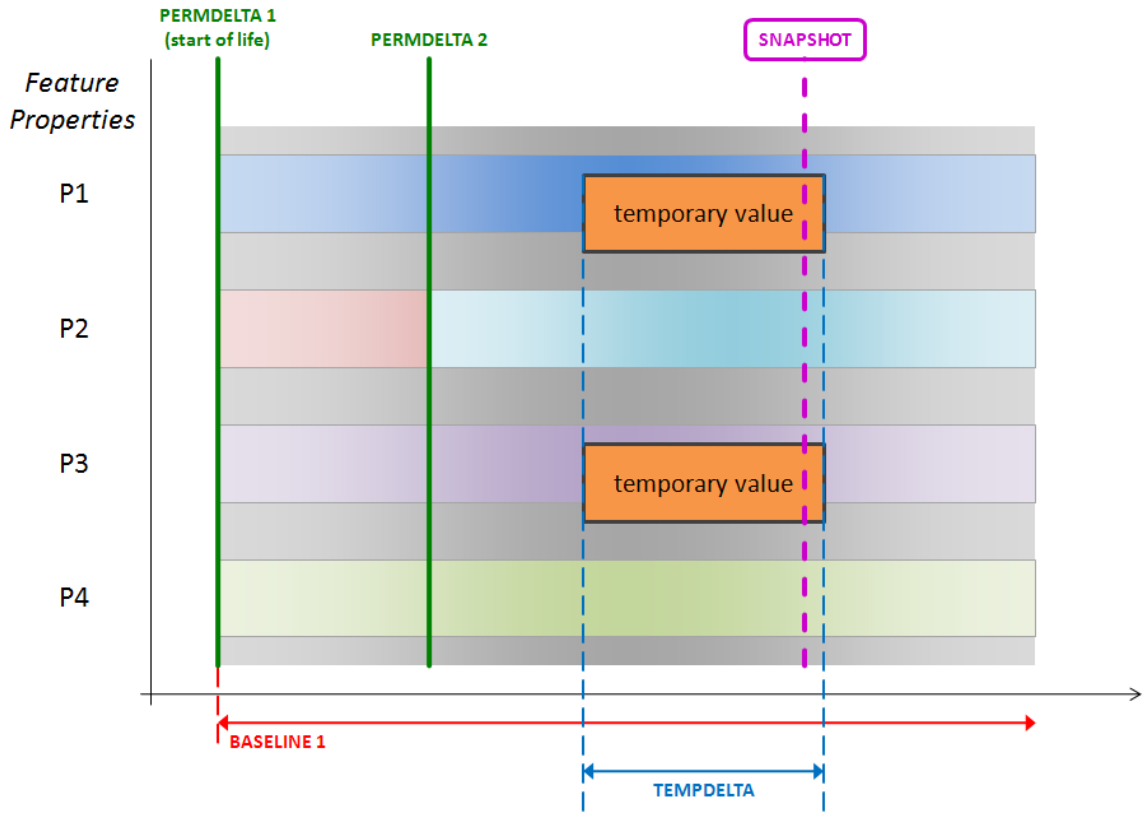

Figure 25 - Snapshot with Baseline, Tempdelta and new Permdelta without according Baseline update

A new Permdelta changes the value of property P2. However, no new Baseline is established to take this event into account and the validity end of Baseline 1 is not updated. This situation is not covered by the current Snapshot definition. It therefore misses an important aspect: the Permdeltas that may occur without according Baselines being established.

10.1.1.2 Recommendation

The definition for Snapshots should be revised to take relevant Permdeltas into account. Relevant Permdeltas are those that have a validTime that is not before or after the validTime of the Baseline that is effective at the time requested for the Snapshot.

Ideally, the order of section 2.5 and 2.6 in the Temporality Model document is switched so that Permdeltas are introduced before Snapshots.

10.1.2 Clarify Snapshot Encoding for Feature Property with Schedule

The Temporality Model does not define in sufficient detail how a PropertyWithSchedule should be encoded in a Snapshot. Two options were discussed during the testbed:

1. Encode the full schedule information for the property.

2. Encode just the applicable value that the PropertyWithSchedule has for the requested point in time, without providing any schedule information.

The following three listings show the difference between these options. Given the following Airspace feature:

Listing 7 – Airspace with activation based on schedule

```
<aixm:Airspace ... gml:id="tmp">
  <gml:identifier
codeSpace="http://www.example.org">i32823jk2823983298</gml:identifier>
  <aixm:timeSlice>
    <aixm:AirspaceTimeSlice gml:id="Airspace1_TS_1">
      <gml:validTime>
        <gml:TimePeriod gml:id="Airspace1_TS_1_TP">
          <gml:beginPosition>2011-01-
01T00:00:00.000Z</gml:beginPosition>
          <gml:endPosition indeterminatePosition="unknown"/>
        </gml:TimePeriod>
      </gml:validTime>
      <aixm:interpretation>BASELINE</aixm:interpretation>
      <aixm:sequenceNumber>1</aixm:sequenceNumber>
      <aixm:correctionNumber>0</aixm:correctionNumber>
      <!-- other feature properties omitted for brevity -->
      <aixm:activation>
        <aixm:AirspaceActivation gml:id="Airspace1_AACT">
          <aixm:timeInterval>
            <aixm:Timesheet gml:id="Airspace1_AACT_TIMESHEET">
              <aixm:timeReference>UTC+2</aixm:timeReference>
              <aixm:startDate>01-01</aixm:startDate>
              <aixm:endDate>31-12</aixm:endDate>
              <aixm:day>ANY</aixm:day>
              <aixm:startTime>00:00</aixm:startTime>
              <aixm:endTime>23:59</aixm:endTime>
            </aixm:Timesheet>
          </aixm:timeInterval>
          <aixm:activity>MILOPS</aixm:activity>
          <aixm:status>AVBL_FOR_ACTIVATION</aixm:status>
          <!-- other AirspaceActivation properties omitted for brevity
-->
        </aixm:AirspaceActivation>
      </aixm:activation>
    </aixm:AirspaceTimeSlice>
  </aixm:timeSlice>
  <aixm:timeSlice>
    <aixm:AirspaceTimeSlice gml:id="Airspace1_TD1">
      <gml:validTime>
        <gml:TimePeriod gml:id="Airspace1_TD1_TP">
          <gml:beginPosition>2011-06-
01T00:00:00.000Z</gml:beginPosition>
          <gml:endPosition>2011-06-05T00:00:00.000Z</gml:endPosition>
        </gml:TimePeriod>
      </gml:validTime>
      <aixm:interpretation>TEMPDELTA</aixm:interpretation>
```

```
      <aixm:sequenceNumber>1</aixm:sequenceNumber>
      <aixm:activation>
        <aixm:AirspaceActivation gml:id="Airspace1_TD1_AACT">
          <!-- no schedule info, thus aixm:status = ACTIVE applies to
the whole validTime of the Tempdelta -->
          <aixm:activity>MILOPS</aixm:activity>
          <aixm:status>ACTIVE</aixm:status>
          <!-- other AirspaceActivation properties omitted for brevity
-->
        </aixm:AirspaceActivation>
      </aixm:activation>
    </aixm:AirspaceTimeSlice>
  </aixm:timeSlice>
  <aixm:timeSlice>
    <aixm:AirspaceTimeSlice gml:id="Airspace1_TD2">
      <gml:validTime>
        <gml:TimePeriod gml:id="Airspace1_TD2_TP">
          <gml:beginPosition>2011-07-
01T00:00:00.000Z</gml:beginPosition>
          <gml:endPosition>2011-07-15T23:59:59Z</gml:endPosition>
        </gml:TimePeriod>
      </gml:validTime>
      <aixm:interpretation>TEMPDELTA</aixm:interpretation>
      <aixm:sequenceNumber>2</aixm:sequenceNumber>
      <aixm:activation>
        <aixm:AirspaceActivation gml:id="Airspace1_TD2_AACT1">
          <aixm:timeInterval>
            <aixm:Timesheet gml:id="Airspace1_TD2_TS1">
              <aixm:timeReference>UTC+2</aixm:timeReference>
              <aixm:day>ANY</aixm:day>
              <aixm:startTime>00:00</aixm:startTime>
              <aixm:endTime>05:59</aixm:endTime>
            </aixm:Timesheet>
          </aixm:timeInterval>
          <aixm:activity>MILOPS</aixm:activity>
          <aixm:status>INACTIVE</aixm:status>
          <!-- other AirspaceActivation properties omitted for brevity
-->
        </aixm:AirspaceActivation>
      </aixm:activation>
      <aixm:activation>
        <aixm:AirspaceActivation gml:id="Airspace1_TD2_AACT2">
          <aixm:timeInterval>
            <aixm:Timesheet gml:id="Airspace1_TD2_TS2">
              <aixm:timeReference>UTC+2</aixm:timeReference>
              <aixm:day>ANY</aixm:day>
              <aixm:startTime>06:00</aixm:startTime>
              <aixm:endTime>17:59</aixm:endTime>
            </aixm:Timesheet>
          </aixm:timeInterval>
          <aixm:activity>MILOPS</aixm:activity>
          <aixm:status>ACTIVE</aixm:status>
          <!-- other AirspaceActivation properties omitted for brevity
-->
        </aixm:AirspaceActivation>
      </aixm:activation>
      <aixm:activation>
```

```
        <aixm:AirspaceActivation gml:id="Airspace1_TD2_AACT3">
          <aixm:timeInterval>
            <aixm:Timesheet gml:id="Airspace1_TD2_TS3">
              <aixm:timeReference>UTC+2</aixm:timeReference>
              <aixm:day>ANY</aixm:day>
              <aixm:startTime>18:00</aixm:startTime>
              <aixm:endTime>23:59</aixm:endTime>
            </aixm:Timesheet>
          </aixm:timeInterval>
          <aixm:activity>MILOPS</aixm:activity>
          <aixm:status>INACTIVE</aixm:status>
          <!-- other AirspaceActivation properties omitted for brevity
-->
        </aixm:AirspaceActivation>
      </aixm:activation>
    </aixm:AirspaceTimeSlice>
  </aixm:timeSlice>
</aixm:Airspace>
```

then the first option for a Snapshot at time 2011-07-02T03:15:00+02:00 would look like this:

Listing 8 – Airspace Snapshot with full schedule

```
<aixm:Airspace ... gml:id="tmp">
  <gml:identifier
codeSpace="http://www.example.org">i32823jk2823983298</gml:identifier>
  <aixm:timeSlice>
    <aixm:AirspaceTimeSlice gml:id="Airspace1_TS">
      <gml:validTime>
        <gml:TimeInstant gml:id="Airspace1_TS_TI">
          <gml:timePosition>2011-07-
02T03:15:00+02:00</gml:timePosition>
        </gml:TimeInstant>
      </gml:validTime>
      <aixm:interpretation>SNAPSHOT</aixm:interpretation>
      <!-- other feature properties omitted for brevity -->
      <aixm:activation>
        <aixm:AirspaceActivation gml:id="Airspace1_TD2_AACT1">
          <aixm:timeInterval>
            <aixm:Timesheet gml:id="Airspace1_TD2_TS1">
              <aixm:timeReference>UTC+2</aixm:timeReference>
              <aixm:day>ANY</aixm:day>
              <aixm:startTime>00:00</aixm:startTime>
              <aixm:endTime>05:59</aixm:endTime>
            </aixm:Timesheet>
          </aixm:timeInterval>
          <aixm:activity>MILOPS</aixm:activity>
          <aixm:status>INACTIVE</aixm:status>
          <!-- other AirspaceActivation properties omitted for brevity
-->
        </aixm:AirspaceActivation>
      </aixm:activation>
      <aixm:activation>
        <aixm:AirspaceActivation gml:id="Airspace1_TD2_AACT2">
```

```
          <aixm:timeInterval>
            <aixm:Timesheet gml:id="Airspace1_TD2_TS2">
              <aixm:timeReference>UTC+2</aixm:timeReference>
              <aixm:day>ANY</aixm:day>
              <aixm:startTime>06:00</aixm:startTime>
              <aixm:endTime>17:59</aixm:endTime>
            </aixm:Timesheet>
          </aixm:timeInterval>
          <aixm:activity>MILOPS</aixm:activity>
          <aixm:status>ACTIVE</aixm:status>
          <!-- other AirspaceActivation properties omitted for brevity
-->
        </aixm:AirspaceActivation>
      </aixm:activation>
      <aixm:activation>
        <aixm:AirspaceActivation gml:id="Airspace1_TD2_AACT3">
          <aixm:timeInterval>
            <aixm:Timesheet gml:id="Airspace1_TD2_TS3">
              <aixm:timeReference>UTC+2</aixm:timeReference>
              <aixm:day>ANY</aixm:day>
              <aixm:startTime>18:00</aixm:startTime>
              <aixm:endTime>23:59</aixm:endTime>
            </aixm:Timesheet>
          </aixm:timeInterval>
          <aixm:activity>MILOPS</aixm:activity>
          <aixm:status>INACTIVE</aixm:status>
          <!-- other AirspaceActivation properties omitted for brevity
-->
        </aixm:AirspaceActivation>
      </aixm:activation>
    </aixm:AirspaceTimeSlice>
  </aixm:timeSlice>
</aixm:Airspace>
```

while the second option for a Snapshot at time 2011-07-02T03:15:00+02:00 would look like this:

Listing 9 – Airspace Snapshot with single value without schedule

```
<aixm:Airspace ... gml:id="tmp">
  <gml:identifier
codeSpace="http://www.example.org">i32823jk2823983298</gml:identifier>
  <aixm:timeSlice>
    <aixm:AirspaceTimeSlice gml:id="Airspace1_TS">
      <gml:validTime>
        <gml:TimeInstant gml:id="Airspace1_TS_TI">
          <gml:timePosition>2011-07-
02T03:15:00+02:00</gml:timePosition>
        </gml:TimeInstant>
      </gml:validTime>
      <aixm:interpretation>SNAPSHOT</aixm:interpretation>
      <!-- other feature properties omitted for brevity -->
      <aixm:activation>
        <aixm:AirspaceActivation gml:id="Airspace1_SN_AACT">
          <aixm:activity>MILOPS</aixm:activity>
```

```
            <aixm:status>INACTIVE</aixm:status>
            <!-- other AirspaceActivation properties omitted for brevity
-->
          </aixm:AirspaceActivation>
        </aixm:activation>
      </aixm:AirspaceTimeSlice>
    </aixm:timeSlice>
</aixm:Airspace>
```

Clarification of encoding PropertiesWithSchedule in Snapshots has already been requested on the AIXM Forum[5]. The request has been accepted and will be incorporated in the next version of the Temporality Model. At the same time, users were invited to suggest further improvements. A summary of the OWS-8 discussions regarding this clarification is therefore provided in this document.

Both options have their advantages and disadvantages - depending on the given use case. The first option is useful for encoding an Extract as a list of Snapshots (see section 10.1.3.4.3). However, simple clients need to evaluate schedules themselves just to find out what the value of a PropertyWithSchedule really is at a given point in time. With the second option, the service that generates the Snapshot is responsible for computing this information and the client can easily read it from the Snapshot. For the encoding of an Extract as a list of Snapshots, the second option causes a significant increase of the size of the resulting representation.

According to the result of the discussion on the AIXM Forum, the second option is preferred and is going to be the default way of encoding a PropertyWithSchedule in a Snapshot. This supports the intent of introducing the Snapshot: supporting simple clients by moving the complex evaluation of timeslices and timesheets to the service.

Supporting the first option - to include full schedule information in Snapshots - can nevertheless be considered as optional service functionality. With option 2. being the default for generating Snapshots, option 1. could be supported via an explicit request.

An additional aspect of encoding a PropertyWithSchedule in a Snapshot is that the value of that property may be undefined at the requested point in time. According to section 2.7 of the Temporality Model, this is possible (but not recommended). If a Snapshot is requested for such a time, then the property shall completely be omitted in the result.

10.1.3 Extract - Extending the Snapshot Concept

10.1.3.1 Introduction

The Temporality Model recognizes the need to communicate the status of an AIXM feature at a given moment in time. In such a situation, it is efficient - and convenient for

---

[5] See the AIXM Forum thread with subject "AIXM Temporality Concept document version 1.0" from October 2010.

the recipient - to encode the information as a feature with a single Snapshot timeslice. The Snapshot contains information on the value of all feature properties for the given time. This information is aggregated from all applicable timeslices (see section 2.5 of the Temporality Model as well as section 10.1.1 and section 10.1.2 in this document for further details).

During OWS-8, the need to get information on the state of an AIXM feature not only for a given point in time but during a given time interval was discussed. This new concept was called *Extract* to better differentiate it from the concept of a Snapshot. The following sections describe the use case for Extracts, how an Extract is computed and how it can be encoded.

10.1.3.2 Use Case

During OWS-8 and previous testbeds, clients subscribed at a standalone Event Service to automatically be notified when updates to aeronautical data were performed. The clients expressed interest in specific updates via filter criteria defined for their subscription.

The AIXM Temporality Model facilitates efficient communication of aeronautical data. Value changes of AIXM feature properties can be modeled and encoded in Permdelta and Tempdelta timeslices which only include the updated properties (together with some timeslice metadata).

However, this efficient encoding is problematic for system entities (like the Event Service) which do not store the complete AIXM feature information themselves.

A subscription filter may for example express interest in activations of airspaces along[6] the route of a (maybe just planned) flight. An AIXM AirspaceActivation does not carry the geographic extent of the airspace concerned (just information on the vertical extent of the activation). An Event Service cannot determine if a new activation event is along the flight route if it receives the Airspace feature with just a timeslice that contains the activation update. The airspace geometry is missing and either has to be added to the update before it is sent to the Event Service or the service must retrieve the geometry itself.

Let us assume that the service retrieves the data from an authoritative datastore itself. Let us further assume that the subscription filter is more complex in that it not only requires knowledge of airspace geometry but of further airspace property values - or values thereof in case of properties with complex type. The service could retrieve the required information for a given point in time - for example the validity start of the activation timeslice - from the data store by requesting snapshots for that time. But how should the service determine which of the possibly changing values of a feature property that is targeted by a filter expression are important? Ideally, the subscriber explicitly states this in the subscription filter. The approach to have an *evaluateDuring* filter function to take

[6] A specific geospatial relationship is not of interest here. What matters is that any spatial relationship can be determined based upon the geometry of the airspace and the flight route.

the dynamic aspect of feature properties for filtering into account was developed in the SAA Pilot and OWS-8 and is documented in [OGC 11-073]. As the properties of an AIXM feature can change their value during the feature lifetime, a user may also need to consider the feature state over a period of time. The evaluateDuring operation thus allows for both the provision of a time instant and time period as parameter for determining which feature data is relevant for filtering. In case that a time instant is provided the Event Service can retrieve a Snapshot of the relevant AIXM feature. Otherwise, it needs to extract the feature information that is relevant for the given period of time.

An “Extract” thus is the equivalent of a Snapshot, just that it provides complete information on the state of a feature for a given period of time, not a point in time. The feature timeslices - Baselines, Permdeltas and Tempdeltas - contain this information. A subset of these timeslices usually already suffices to represent the requested information. Thus it is unnecessary to retrieve the complete feature information from the authoritative data store.

In addition to the “Eventing” use case just described, another use of an “Extract” may be caching. A client may want to explore the state of an AIXM feature for given point in time but the application already automatically retrieves feature state information for adjacent periods of time in a background process. This would speed up browsing through the data and is similar to mapping clients which cache map tiles that are adjacent to the current view as well as zoom level.

An approach for retrieving an AIXM feature with just the desired subset of timeslices is documented in [OGC 11-073]. The following sections describe which timeslices are needed for an Extract and the options for encoding it.

10.1.3.3 Computation

An Extract is computed by identifying the timeslices of an AIXM feature that are relevant for the time of interest assigned to the Extract, i.e. the time period for which the state of an AIXM feature shall be determined. It is tempting to say that only those timeslices are relevant that are not before and not after the time of interest. In addition, one might think that only Baselines and Tempdeltas are of interest. Although both assumptions cover most cases, they are not entirely sufficient. Consider the following figure.

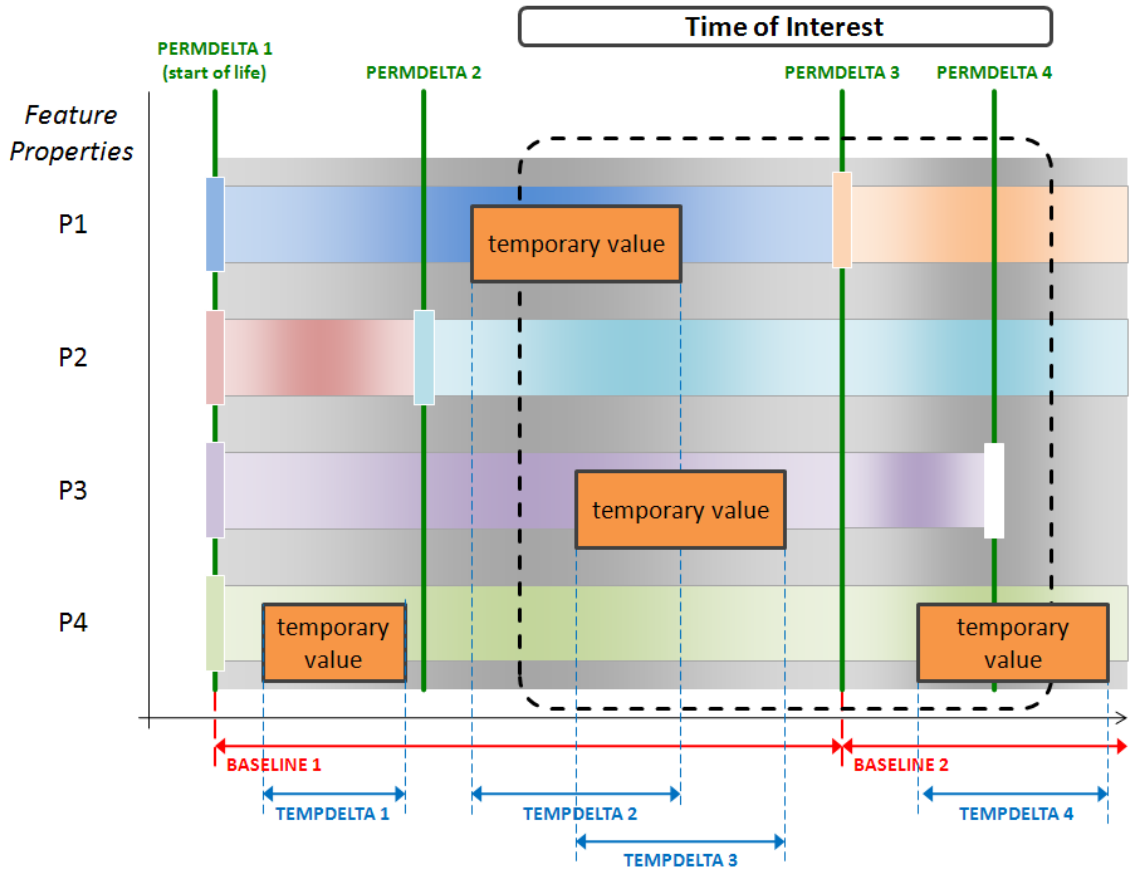

Figure 26 – State of an AIXM feature defined via its timeslices and time of interest of an Extract of that feature

The Temporality Model allows that Permdeltas are not accompanied by an according Baseline - see section 10.1.1. In Figure 26, Permdelta two and four fall into this category. They permanently change the values of properties P2 and P3 respectively. Permdeltas therefore need to be taken into account as well for computing an Extract.

The following list defines which timeslices are relevant for an Extract:

1. Baselines which are not before and not after the time of interest of the Extract are relevant.
2. Permdeltas which are not after the time of interest of the Extract are relevant, unless:
    a. they have a valid time which equals the start of the valid time of one of the Baselines identified in the previous step (in that case, the Baseline incorporates the state change that such a Permdelta represents)
    b. a Baseline exists whose valid time is not before or after the start of the time of interest of the Extract and the valid time of the Permdelta is before the valid time of that Baseline (the Baseline overrides or incorporates all state changes that such Permdeltas represent)

3. Tempdeltas which are not before and not after the time of interest of the Extract are relevant.

The following notes apply:

- Timeslices that have been corrected by another one are irrelevant for the Extract.
- Timeslices that have been cancelled (see Temporality Model section 3.6 for further details) are irrelevant for the Extract.
- A further optimization by detecting Tempdeltas whose state changes are overwritten through Tempdeltas with higher sequence number (see Temporality Model section 3.8 “Overlapping TimeSlices and corrections” for further details) would be complex and is thus not considered in the identification of relevant timeslices. The optimization would need to determine that the temporary state changes communicated by a given Tempdelta are completely - i.e., for the whole valid time of that Tempdelta - superseded through Tempdeltas with higher sequence number. The possible stacking of such Tempdeltas complicates matters, as well as the fact that a Tempdelta may contain temporary values for more than one AIXM feature property.

The following figure shows which timeslices are relevant for the extract in the example shown in Figure 26.

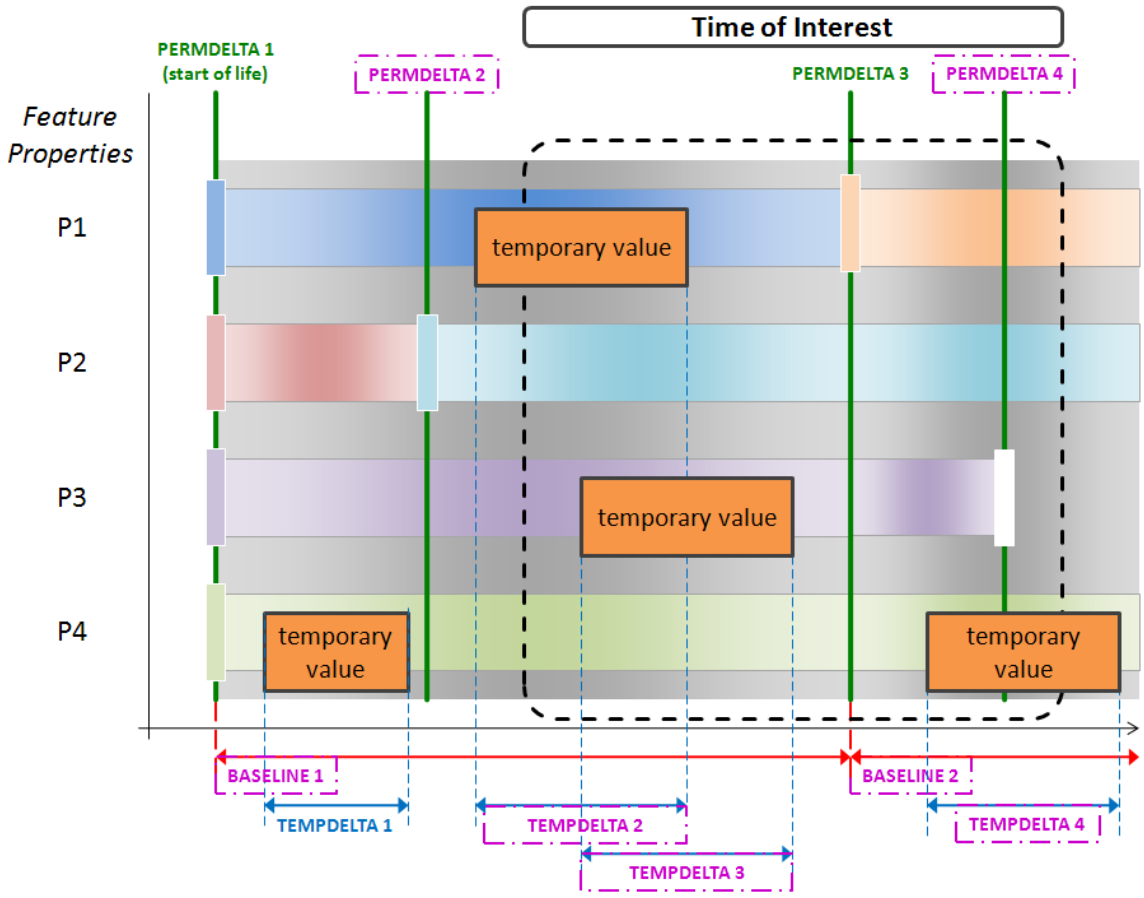

Figure 27 –State of an AIXM feature defined via its timeslices - timeslices relevant for the Extract with given time of interest are depicted in purple.

The previous figures show an AIXM feature which is defined for the whole time of interest of an Extract. In the general case, one or more property of an AIXM feature - even the whole feature - may not be defined for that time. Therefore, information on the feature state may be missing - partially or completely - for an Extract. An example is shown in the following figure. The encoding of an Extract needs to take this into account.

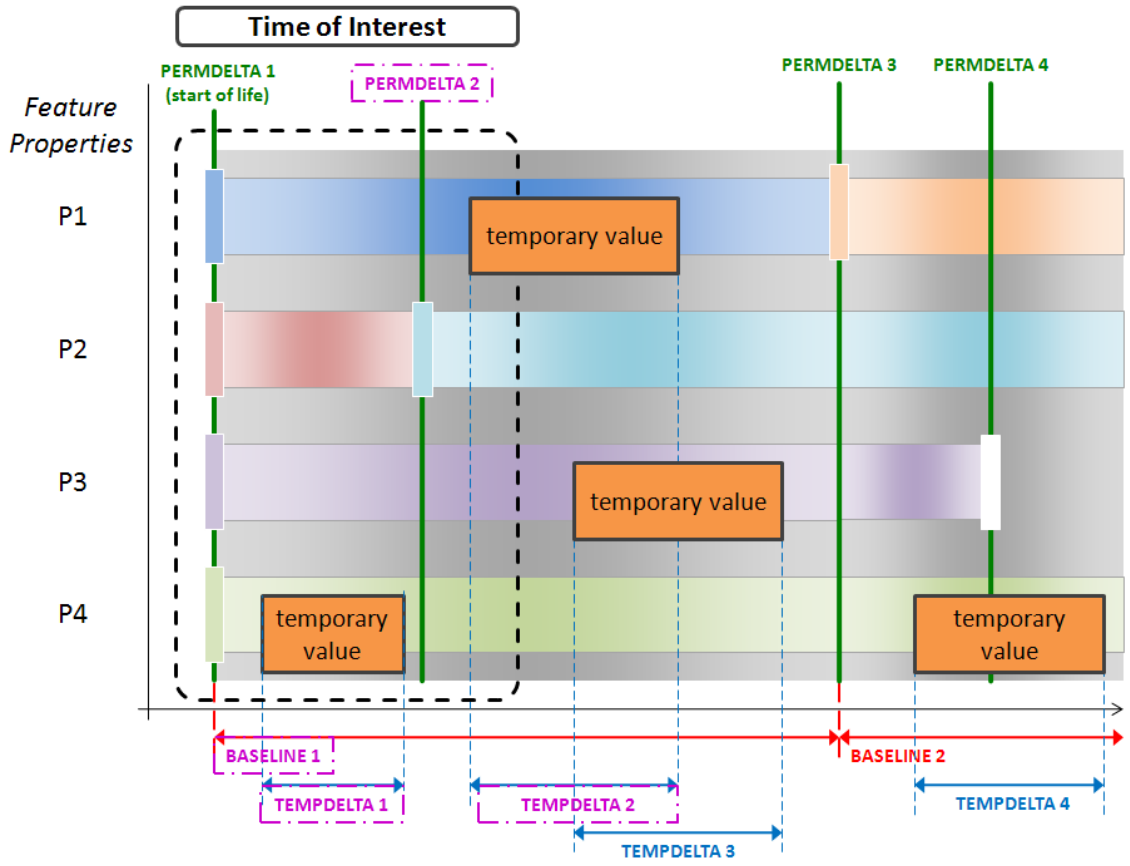

Figure 28 –Timeslices relevant for an Extract (depicted in purple) whose time of interest is partially outside the feature lifetime.

Note that an Extract contains information on the feature state only for its defined time of interest and neither the time before or after that period of time. Figure 28 exemplifies this. There, Tempdelta 2 is of relevance to the Extract, but Tempdelta 3 is not as it is after the time of interest. Only the information in Tempdelta 2 is encoded in the extract, not that of Tempdelta 3. It is important to understand that any changes that occur after the time of interest are not considered by the computation instructions defined in this section. An Extract therefore does not enable clients to make any assumption on the state of an AIXM feature before or after the time of interest.

10.1.3.4 Encoding

Once the timeslices relevant for an Extract have been identified, the information on the feature state that these timeslices represent needs to be encoded so that it can be communicated to a client. Two options were discussed during the testbed. They are described in the following sections.

10.1.3.4.1 Snapshot List

The state of a feature at a given point in time can be represented by a Snapshot timeslice. An Extract contains information on the state of an AIXM feature for a period of time and

thus the feature state may change during that time. A Snapshot timeslice as defined by the Temporality Model has a validTime that is represented by a time instant. However, the state of a feature usually does not change so frequently that the resolution of system clocks would not be able to represent each change as a different point in time. The state of an AIXM feature therefore is most often constant over a certain period of time - which can last seconds, minutes, hours or days, depending on the scenario. This period of time in which the feature state is constant can be represented in the validTime of a Snapshot. An Extract can thus be represented by a list of these Snapshots.

From the list of relevant timeslices, an application needs to compute the feature state at the start and end of the time of interest as well as all state changes in between. For the example shown in Figure 27, a list of seven Snapshots (see the following figure) would be used to encode the information of the relevant timeslices.

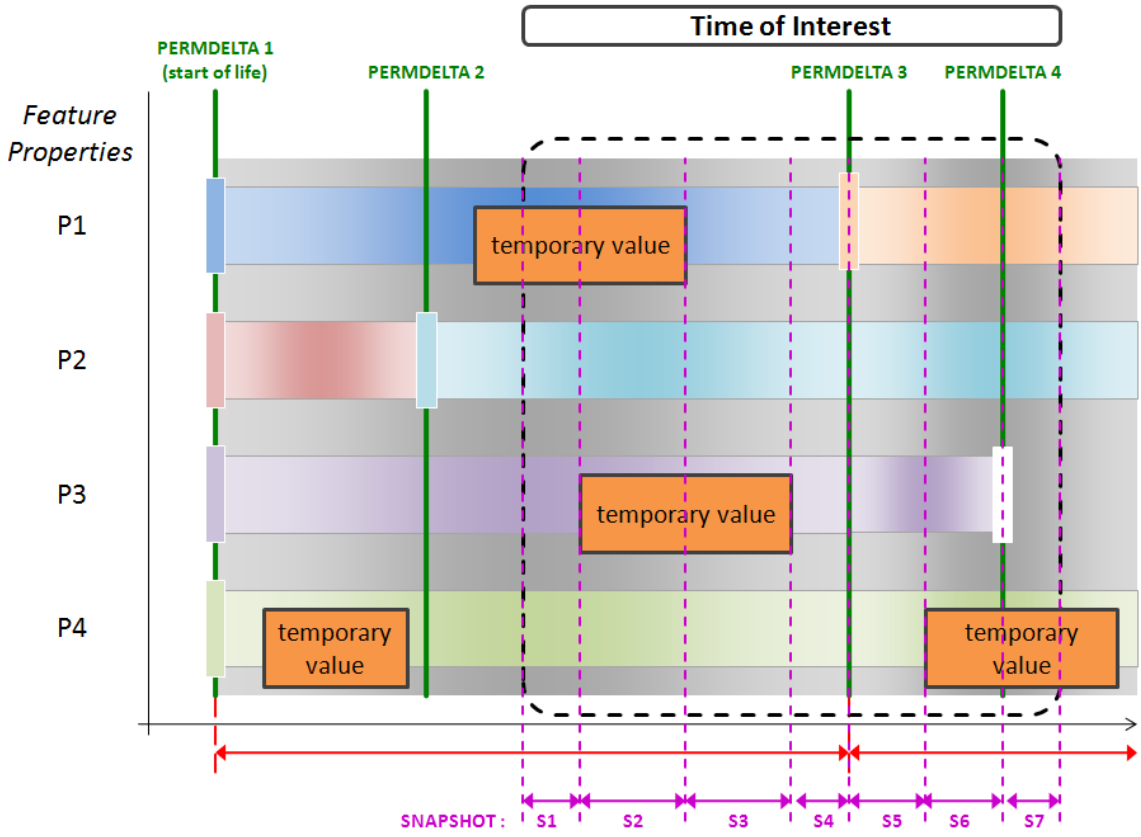

Figure 29 – Extract whose time of interest is during the feature lifetime - encoded as a list of Snapshots

The validTime of the first/last Snapshot begins/ends at the start/end of the time of interest (if the feature state is defined for these points in time). This takes into account that an Extract does not provide information about the state of an AIXM feature before or after the time of interest.

As discussed before, the time of interest of an Extract may not be contained within the lifetime of an AIXM feature. Accordingly, the list of Snapshots may not cover the whole time of interest or may even be empty[7]. For the example shown in Figure 28, a list of Snapshots as shown in the following figure would be created.

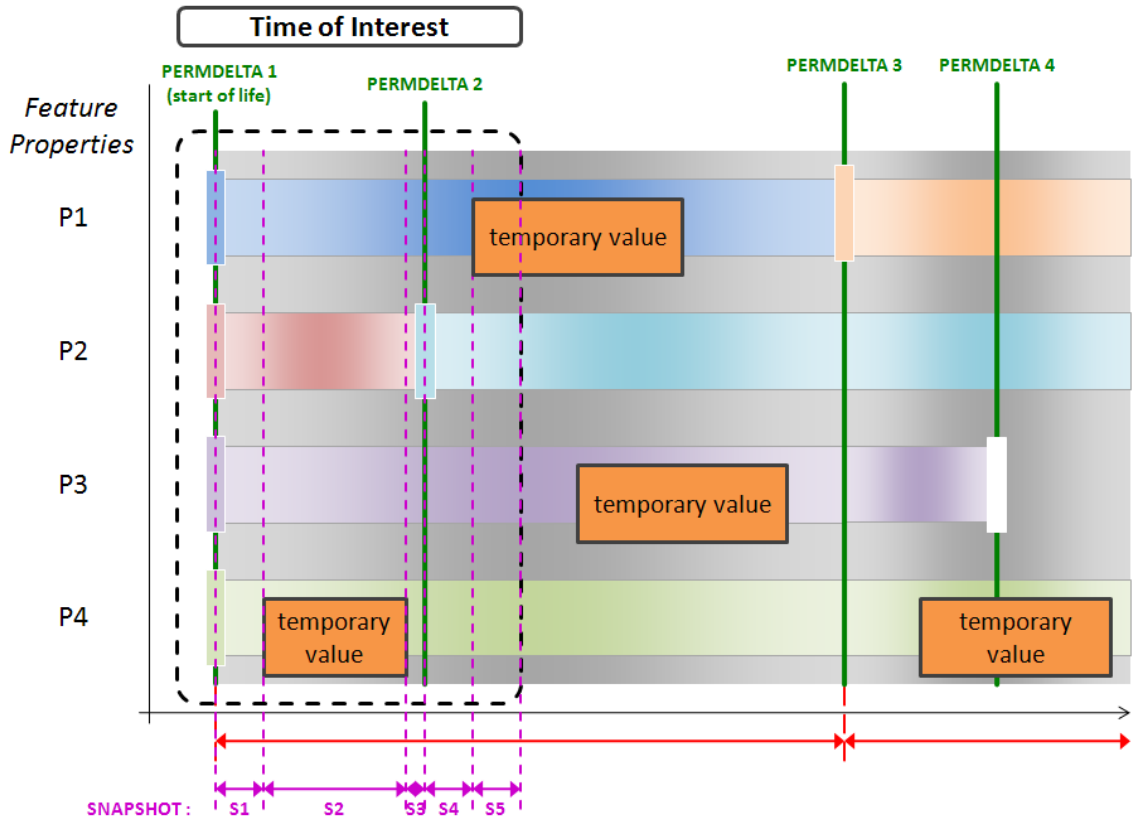

Figure 30 – Extract whose time of interest overlaps the feature lifetime - encoded as a list of Snapshots

The start of the validTime of Snapshot S1 equals the beginning of the feature lifetime, taking into account that the feature state is undefined before that point in time.

10.1.3.4.2 Feature Subset

Once the relevant timeslices for an Extract have been identified, a straightforward way to encode them is to simply add them to the feature and to return it.

This automatically takes into account the situations in which the feature state is undefined at the beginning/end of the time of interest. If the feature is completely undefined for the time of interest, then no timeslices are found for the Extract and the feature should completely be omitted in the response.

---

[7] The XML encoding of AIXM feature has to contain at least one timeslice. Therefore, the response to a request of a feature Snapshot that is outside the lifetime of the feature should omit the feature altogether.

Note that even though some timeslices may be valid before or after the time of interest of the Extract (as explained in section 10.1.3.3), the information from these timeslices does not necessarily suffice to represent the complete feature state during the validTime of these timeslices. For example, in Figure 28 Tempdelta 3 is not a relevant timeslice but changes the feature state during the validTime of Tempdelta 2.

10.1.3.4.3 Comparison

Both encoding approaches are of interest to clients.

Simple clients that usually consume Snapshots - for example to monitor state changes (like traffic management and flight plan processing systems as mentioned in the Temporality Model) - may prefer to receive a list of Snapshots. This can be especially useful in situations in which such a system did not receive state change information for a certain period of time, for example because it was offline. In that case the system can request an Extract for that period of time and thus get the Snapshots that it missed.

Clients that support dynamic features according to the Temporality Model and thus represent and determine feature state based upon Baselines, Permdeltas and Tempdeltas will prefer an Extract that is encoded via the relevant timeslices. The example in which a client did not receive information on state changes of a feature is applicable here as well. Such changes may usually have been communicated according to the Digitial NOTAM specification. Missing information can be retrieved from the authoritative datastore as an Extract and the resulting timeslices directly incorporated into the client's internal representation of the feature.

The efficiency of both encoding approaches with respect to the size of the resulting representation has not been investigated in detail. However, some relevant aspects were discussed.

- Snapshots contain values for all feature properties (unless they are undefined at/during the Snapshot validTime). Feature property values that do not change in two consecutive Snapshots will still need to be represented in a Snapshot list. Referencing property values via xlink:href may be used to reduce the amount of data that is caused by the repeated encoding of such unchanging property values.
- In case that a client wanted to find out which information actually changed between two consecutive snapshots - which is different to just determining the value that is applicable at a given point in time - it would have to compare the whole list of properties. Usage of xlink:href may help in this task as a client can more easily determine if the values of a feature property from two Snapshots refer to the same object.
- The number of Snapshots used to encode an Extract should not be significantly higher than the number of the relevant timeslices. However, if the value of a PropertyWithSchedule was encoded as multiple Snapshots - see section 10.1.2 for further details - then the approach to encode an Extract as a list of Snapshots is

less efficient. Figure 31 illustrates the issue. There, the value of property P3 is changed temporarily, based on a schedule. The number of Snapshots required to encode the complete feature state is significantly higher than in the situation without changes by schedule (see Figure 29). The situation would get worse in case of different properties with overlapping schedules. Allowing Snapshots with property values defined by schedule would avoid this issue.

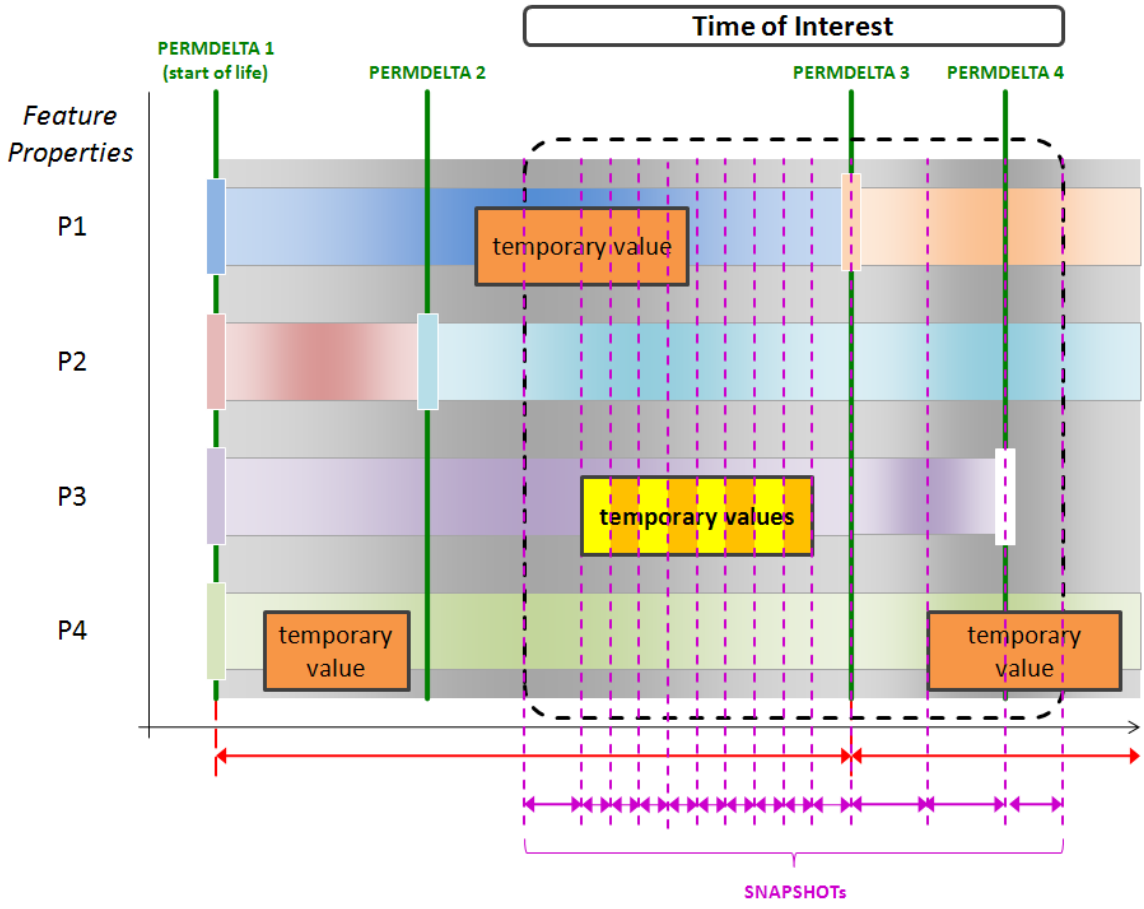

Figure 31 – Extract including temporary changes by schedule - encoded as a list of Snapshots

10.1.3.5 Summary

The Extract is a logical extension of the Snapshot concept defined in the Temporality Model. It provides information on the full state of a feature not only at a given point in time but during a period of time. This is useful in situations in which the full history of a feature is not required, instead only a well defined period of time is of interest for computations (e.g. filtering and portrayal). Instructions for computing an Extract by identifying the relevant timeslices have been provided in section 10.1.3.3. Two approaches for encoding an Extract have been explained and discussed - see section 10.1.3.4.

10.1.4 Reconsider Rules for Handling Changes to Multi Occurring Properties

Section 3.5 in the AIXM Temporality Model explains how multi occurring properties should be handled when one or more of their values changes (permanently or temporarily). The issues that the section addresses is that simple UML <<object>>s are not identifiable (in the XML encoding they do not have a gml:identifier). Recognizing which value of a multi-occurring property changed is not possible. Therefore, the Temporality Model states that "*in a PERMDELTA or TEMPDELTA Time Slice, multi-occurring properties shall be provided with all occurrences included*". This statement does not exactly define which value occurrences shall be included, which is likely due to the simplicity of the chosen example. Section 3.5 appears to only cover properties without schedule. Furthermore, it does not appear to take into account a specific edge case. This is problematic as will be explained in the following paragraphs.

10.1.4.1 Delta for multi-occurring property with schedule

Let us consider the case of an Airspace which can be activated at certain times. An "activation" is a multi occurring property of an Airspace that has a schedule (and thus is valid for certain time(s)). Furthermore, it is a simple <<object>>, not a <<feature>>. Therefore, strictly following the rule from the Temporality Model section 3.5, whenever a change in the activation schedule (either changing the regular schedule or performing an exceptional activation - such as when some military activity is taking place) of the Airspace occurs, the delta timeslice would need to incorporate all activation objects from the applicable static information, even those whose schedule is before or after the valid time of the new activation.

This is not necessarily an issue in situations where a permanent change is communicated. In such a situation, all activations that were applicable in the past can be omitted in the new Permdelta/Baseline. These activations would still be stored in the history of timeslices of the Airspace feature and thus can be queried if necessary. Furthermore, permanent changes in Airspace activations usually concern a change of the regular schedule and do not incorporate exceptional activations. Exceptional activations are communicated via Tempdeltas (see the Digital NOTAM Event Specification for further details). For exceptional activations, however, the rule for multi-occurring properties as it is currently defined in the Temporality Model is not ideal.

If the status of an exceptional activation is constant throughout the valid time of a Tempdelta then that Tempdelta would only need a single activation value. However, according to the current rule, it looks like the Tempdelta needs to include all other Airspace/activation values that are contained in Permdeltas/Baselines that are not before and not after the valid time of the Tempdelta. This definitely would be unnecessary overhead as in this situation a single activation object suffices to fully define the state of the activation property throughout the valid time of the Tempdelta.

10.1.4.2 Temporary change of multi-occurring property that overlaps a permanent change

A Permdelta that changes the value of a multi occurring AIXM feature property may have a valid time that is during the valid time of the Tempdelta - see the following figure.

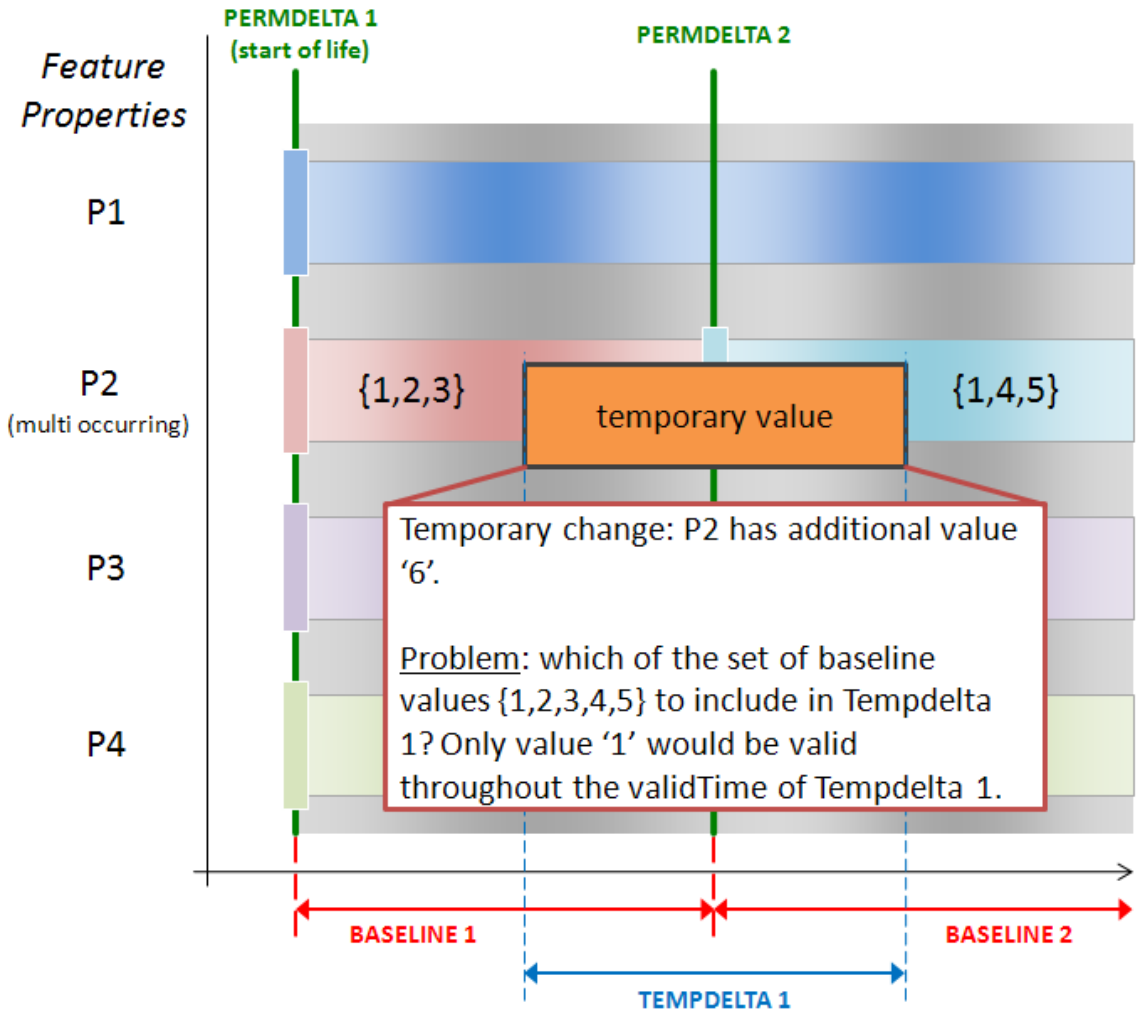

Figure 32 – Permanent change of a multi occurring feature property occurring during the valid time of a temporary change of that property

Here, property P2 is multi occurring. Its static set of values ({1,2,3} in Baseline 1) is changed permanently (via Permdelta 2 to {1,4,5}). However, a temporary change is applied to the property (via Tempdelta 1 – adding value '6') and its valid time contains the valid time of the permanent change.

This situation is not explicitly described in the AIXM Temporality Model. As it is not forbidden, it may occur[8]. In such a situation it does not matter whether the multi occurring property (P2) does or does not have a schedule - either way it is unclear how the Tempdelta should incorporate the unchanged property values from the static information (Baseline 1 and 2). Should all unchanged values be incorporated? As the set of values for P2 changes permanently via Permdelta 2, the values that would be incorporated from the Baselines 1 and 2 in Tempdelta 1 following the current rule from

[8] If this situation is not allowed to happen then the AIXM-TM should explicitly state this.

the Temporality Model would not be valid throughout its validTime (in Figure 32, only value'1' would be valid throughout the validTime of Tempdelta 1) – although that should be the contract for a Tempdelta. Furthermore, the value of P2 in Tempdelta 1 does not even need to contain such values as long as the value of P2 is fully defined throughout the valid time of P2 (see Temporality Model section 2.7 for further details).

During OWS-8, different options for solving this issue were developed, depending on whether the multi-occurring property is or is not a *PropertyWithSchedule* (again, see Temporality Model section 2.7 for further details).

If P2 is a property without schedule, then a solution is to model the temporary change via two Tempdeltas as shown in the following figure.

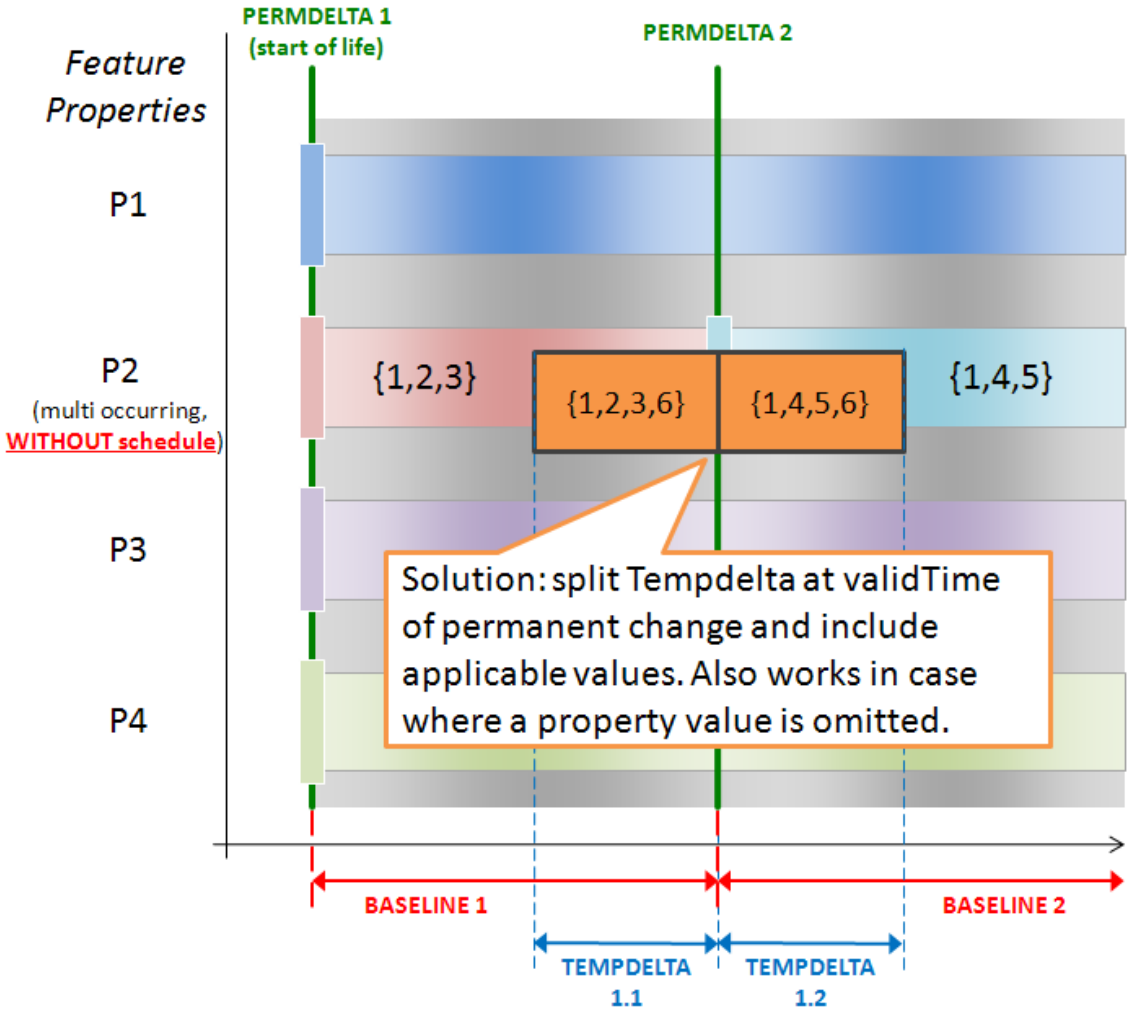

Figure 33 – Handling the delta overlap issue for a multi-occurring property without schedule

As we can see, the validTime of the Permdelta equals the end time of Tempdelta 1.1 and the start time of Tempdelta 1.2. By splitting the temporary change, the information can easily be merged with the baseline information that is still valid.

If P2 is a property with schedule, then there are two options for solving the issue – see Figure 34.

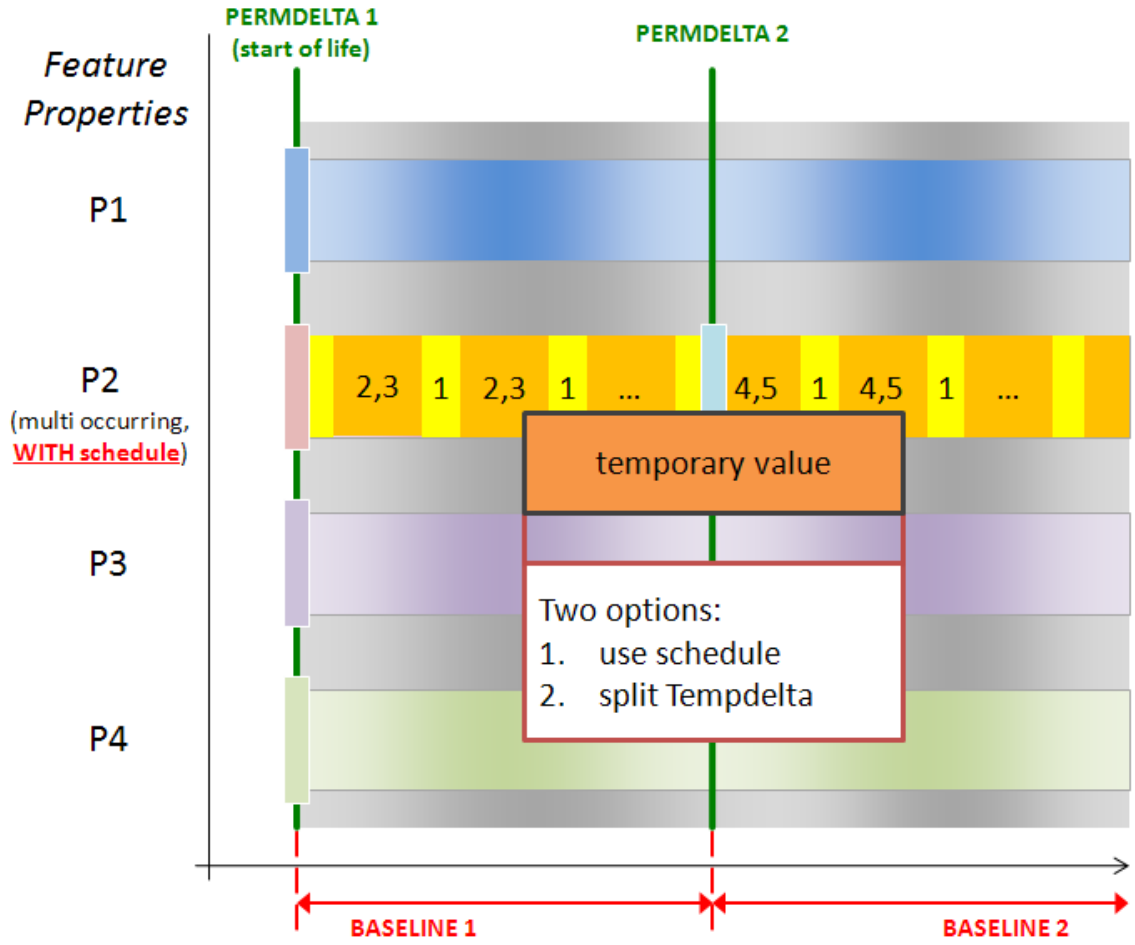

Figure 34 – Options for solving the delta overlap issue for a multi-occurring property with schedule

In this example, the schedule of P2 in Baseline 1 defines that the set of valid values for P2 alternates between {1} and {2,3}. The permanent change introduced by Permdelta 2 changes the schedule so that the set of valid values for P2 alternate between {1} and {4,5}. The temporary change would now add the value '6' to the set of valid values – for the purpose of this example let us assume that this value is not constant but valid at the same time that the value sets {2,3} and {4,5} are valid. This can be modeled and encoded in two ways. On the one hand, the schedule can be adapted to take the additional value into account (see Figure 35). On the other hand, the temporary change can again be modeled via two distinct Tempdeltas (see Figure 36).

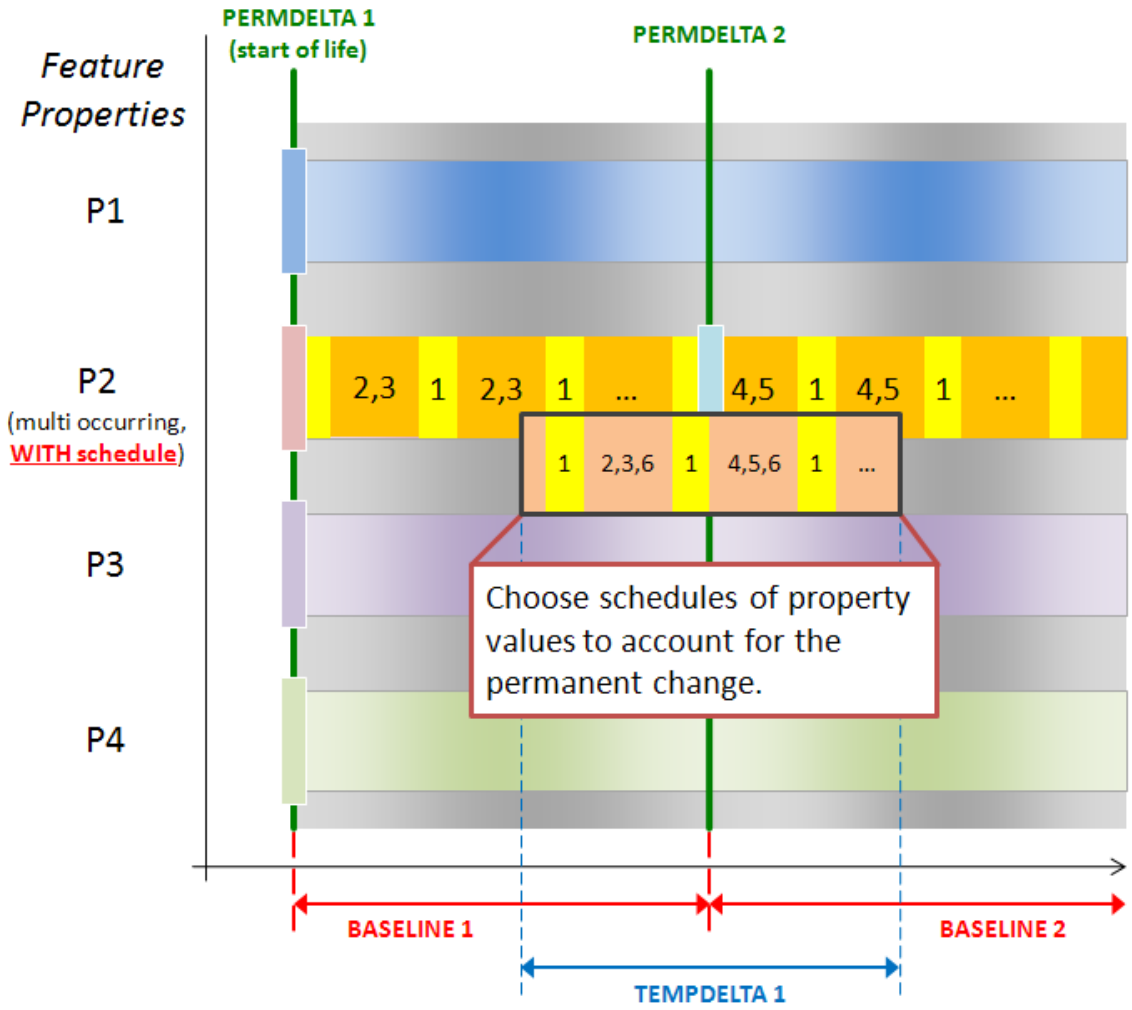

Figure 35 – Solving the delta overlap issue for a multi-occurring property with schedule via the schedule itself

By appropriately encoding the schedule via property values with according timesheets, the valid sets of values for P2 can correctly be represented throughout the validTime of a single Tempdelta.

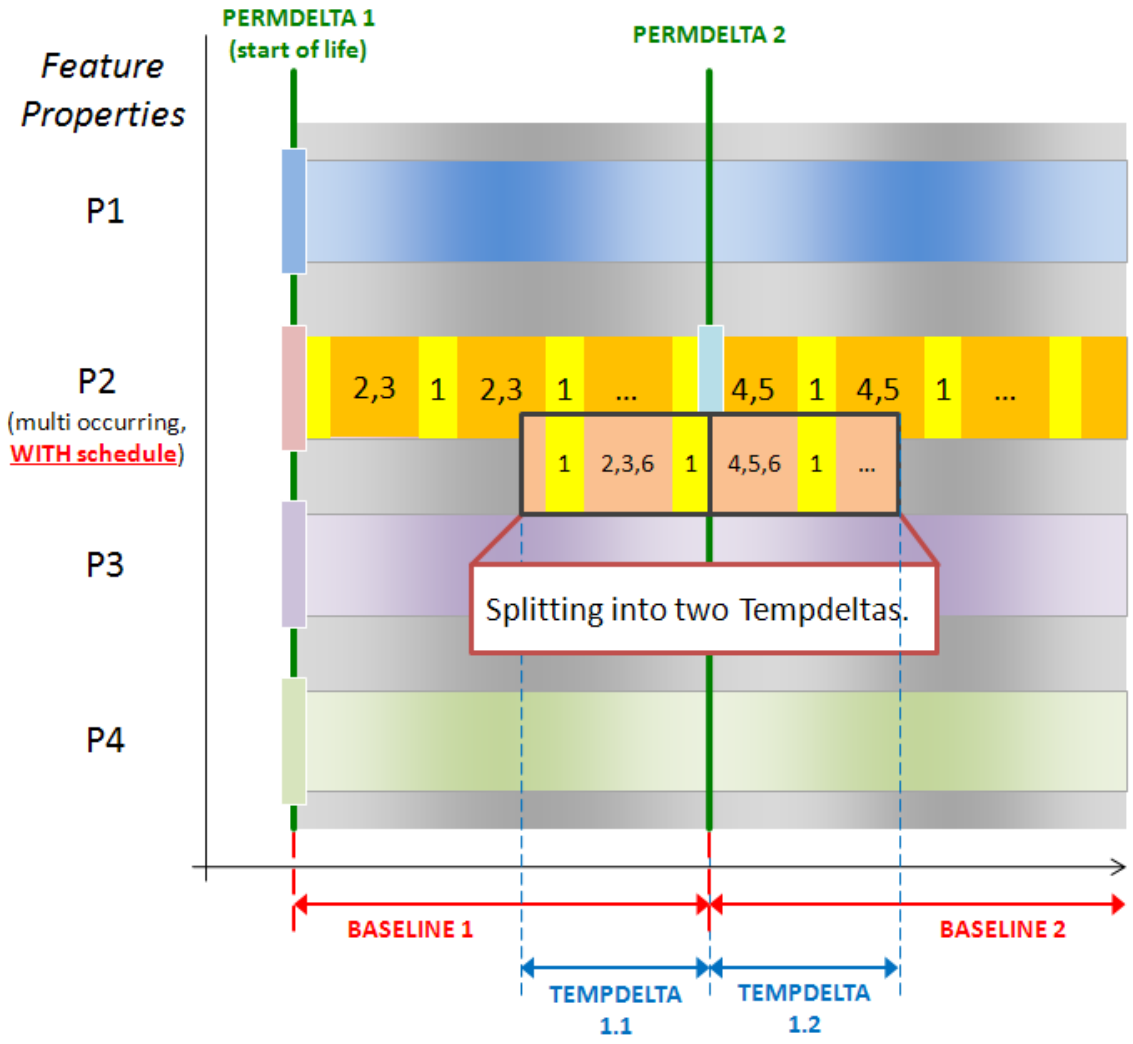

Figure 36 – Solving the delta overlap issue for a multi-occurring property with schedule by encoding the temporary change via two Tempdeltas

10.1.4.3 Recommendation

Section 3.5 in the Temporality Model should be revised. On the one hand, it needs to take changes for multi-occurring properties with schedule into account. On the other hand, it needs to account for the edge case of a permanent change to a multi occurring property that occurs during a temporary change of that property. For the latter, a recommendation to handle the situation by encoding two distinct Tempdeltas should be added. This appears to be the easiest solution which works for multi-occurring properties with and without schedule. However, being able to model the temporary change as a single Tempdelta may be useful for multi-occurring properties with schedule. This should therefore not be prohibited by the Temporality Model.

10.1.5 Incorporate Extension Property Handling

During the FAA SAA Dissemination Pilot (see OGC 11-055) the handling of AIXM feature and object extension properties was discussed. More specifically, the correct handling according to the rules defined in the AIXM Temporality Model. The discussion quickly revealed that the Temporality Model itself does not clearly specify how extension properties are to be handled. This section summarizes the result of the discussion. The Temporality Model should be revised to incorporate the instructions for handling AIXM extension properties.

The following diagram shows an extension of the AIXM 5.1 Airspace feature that was used to add Special Use Airspace (SUA) information to that feature type in the SAA Pilot.

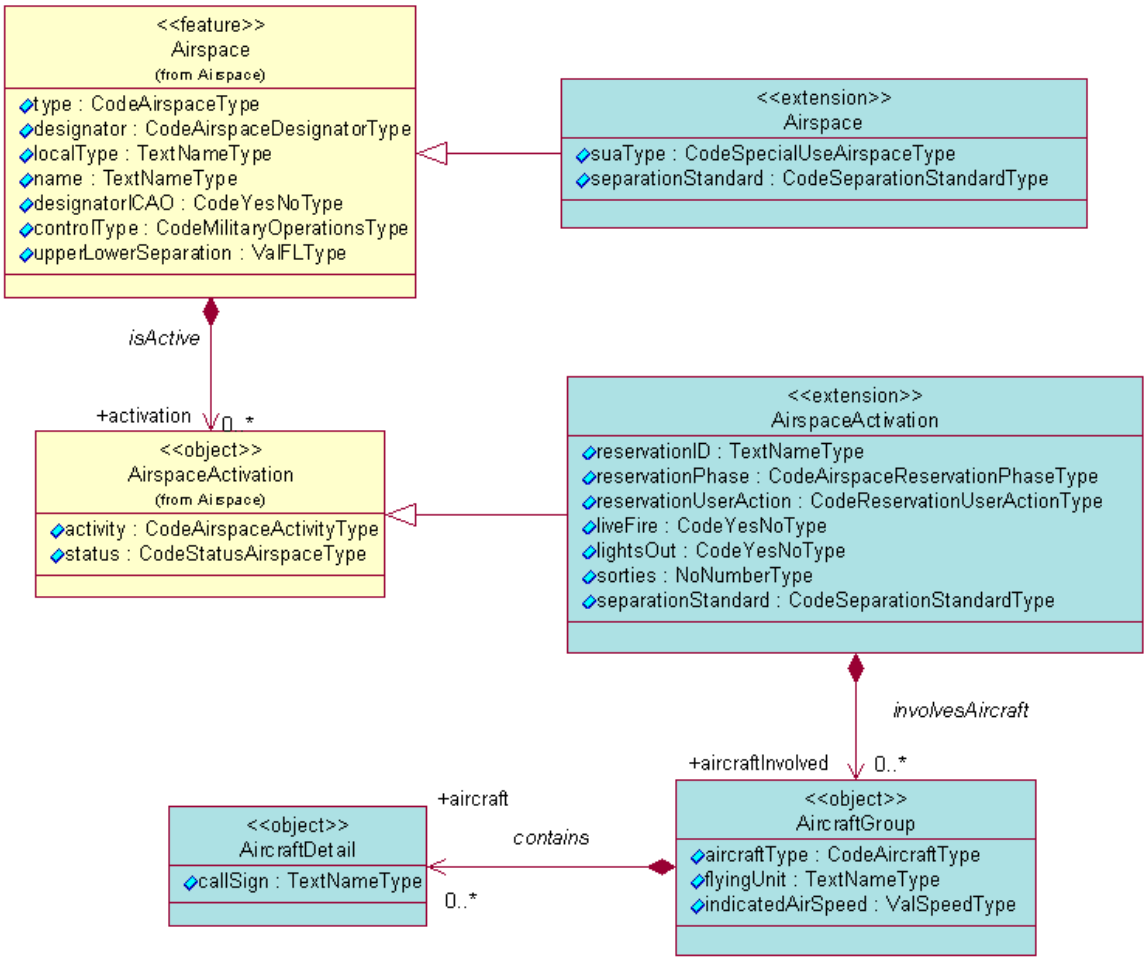

Figure 37 – Special Use Airspace Feature extension (for AIXM 5.1)

Another extension (shown in the following figure) adds Special Activity Airspace (SAA) information to an AIXM 5.1 Airspace feature.

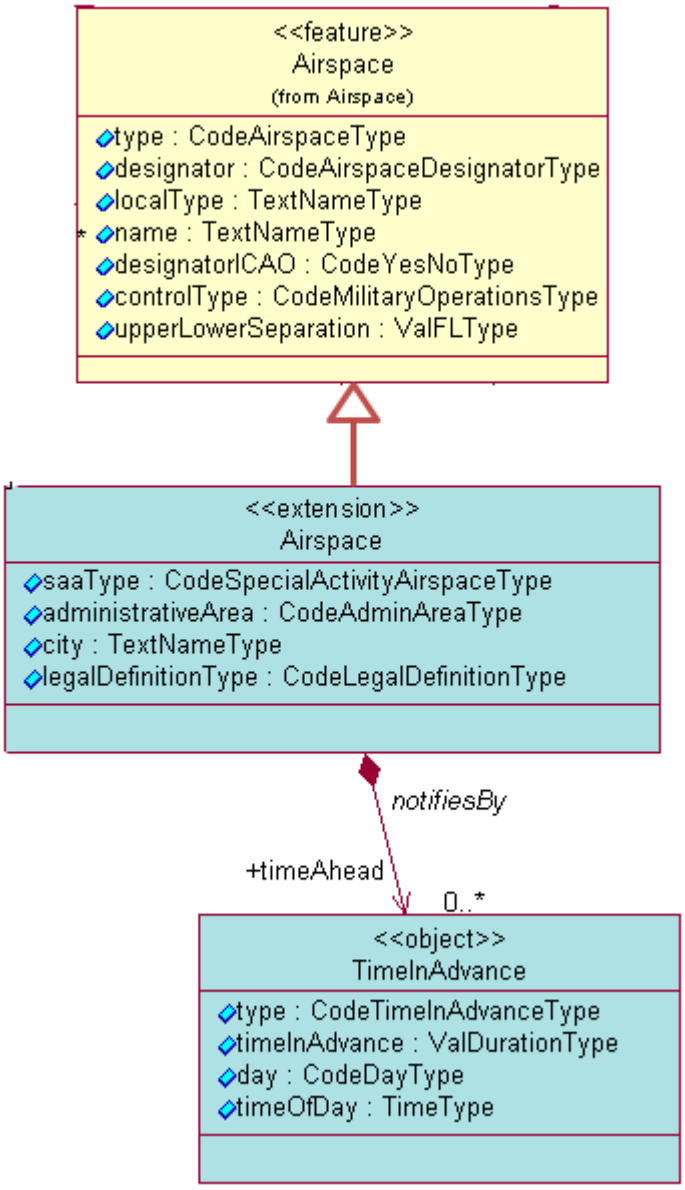

Figure 38 – Special Activity Airspace extension (for AIXM 5.1)

The extensions add additional properties to the Airspace feature. We will use these extension models in the examples of this section.

The following listing contains an XML example of an AIXM Airspace Baseline timeslice that incorporates extension values from both extension models. Note that each extension value is a complex AirspaceExtension element within its own namespace.

Listing 10 – AIXM Airspace Baseline with both SUA and SAA extension elements

```
<aixm:Airspace gml:id="xyz_0"
xmlns:aixm="http://www.aixm.aero/schema/5.1"
xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:sua="urn:us:gov:dot:faa:aim:saa:sua:5.1"
xmlns:saa="urn:us:gov:dot:faa:aim:saa:5.1">
      <gml:identifier
codeSpace="http://www.faa.gov/nasr">xyz</gml:identifier>
      <aixm:timeSlice>
        <aixm:AirspaceTimeSlice gml:id="urn.uuid.xyz_1">
          <gml:validTime>
            <gml:TimePeriod gml:id="urn.uuid.xyz_2">
              <gml:beginPosition>2011-03-10-05:00</gml:beginPosition>
              <gml:endPosition
indeterminatePosition="unknown"></gml:endPosition>
            </gml:TimePeriod>
          </gml:validTime>
          <aixm:interpretation>BASELINE</aixm:interpretation>
          <aixm:sequenceNumber>1</aixm:sequenceNumber>
        <!-- other AIXM Airspace feature properties omitted for brevity
-->
          <aixm:extension>
            <saa:AirspaceExtension gml:id="urn.uuid.xyz_10">
              <saa:saaType>SUA</saa:saaType>
              <saa:administrativeArea>NORTH
CAROLINA</saa:administrativeArea>
              <saa:city xsi:nil="true" nilReason="inapplicable"/>
              <saa:legalDefinitionType>HUMAN</saa:legalDefinitionType>
              <saa:timeAhead xsi:nil="true" nilReason="inapplicable"/>
            </saa:AirspaceExtension>
          </aixm:extension>
          <aixm:extension>
            <sua:AirspaceExtension gml:id="urn.uuid.xyz_11">
              <sua:suaType>RA</sua:suaType>

<sua:separationStandard>UNSPECIFIED</sua:separationStandard>
            </sua:AirspaceExtension>
          </aixm:extension>
        </aixm:AirspaceTimeSlice>
      </aixm:timeSlice>
    </aixm:Airspace>
```

The following rules apply for handling changes to extension properties.

AIXM feature extensions do not follow the rule for multi-occurring properties as defined in section 3.5 of the Temporality Model.

Explanation

The purpose of the rule for multi-occurring feature properties in delta timeslices was to avoid ambiguity. Following the example in section 3.5 of the Temporality Model, if an AirportHeliport had several cityServed in its Baseline and upon a change of this property just the new value was put into the delta timeslice, then it would be unclear which ones from the Baseline remain valid. This is not the case for AIXM feature extensions, because the child elements of each <aixm:extension> always have a different namespace (event, sua, saa, etc.) and there can only be a single extension element per namespace. This is implicitly defined by the rules for extending AIXM features/objects as detailed in the "AIXM - Application Schema Generation" and "AIXM - UML to XML Schema Mapping" documents.

Note: that there can only be one extension element per XML namespace (assigned to a specific AIXM extension model, for example the SUA or SAA extensions) and per AIXM feature/object should be explicitly stated in both the "AIXM - Application Schema Generation" and "AIXM - UML to XML Schema Mapping" documents.

Therefore, there is no risk for ambiguity and we can safely say that an AIXM feature extension can be excepted from the rule for multi occurring feature properties. Each extension is semantically different, it does not make sense to treat them as normal multi occurring feature properties, because they are not.

The advantage of this approach is that a value change in an AIXM feature extension does not require all unchanged extensions from the baseline data (and from different extension models) to be included in the timeslice that represents the change.

A Permdelta that changes the SUA type for the Airspace example from Listing 10 therefore does not need to include the saa:AirspaceExtension. The following listing shows an example of the resulting Permdelta.

Listing 11 – AIXM Airspace Permdelta with changed SUA extension element

```
<aixm:Airspace gml:id="xyz_0"
xmlns:aixm="http://www.aixm.aero/schema/5.1"
xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:sua="urn:us:gov:dot:faa:aim:saa:sua:5.1"
xmlns:saa="urn:us:gov:dot:faa:aim:saa:5.1">
      <gml:identifier
codeSpace="http://www.faa.gov/nasr">xyz</gml:identifier>
      <aixm:timeSlice>
        <aixm:AirspaceTimeSlice gml:id="urn.uuid.xyz_1">
          <gml:validTime>
            <gml:TimePeriod gml:id="urn.uuid.xyz_2">
              <gml:beginPosition>2011-03-15-05:00</gml:beginPosition>
              <gml:endPosition
indeterminatePosition="unknown"></gml:endPosition>
            </gml:TimePeriod>
          </gml:validTime>
          <aixm:interpretation>PERMDELTA</aixm:interpretation>
          <aixm:sequenceNumber>1</aixm:sequenceNumber>
          <aixm:extension>
            <sua:AirspaceExtension gml:id="urn.uuid.xyz_11">
              <sua:suaType>WA</sua:suaType>

<sua:separationStandard>UNSPECIFIED</sua:separationStandard>
            </sua:AirspaceExtension>
          </aixm:extension>
        </aixm:AirspaceTimeSlice>
      </aixm:timeSlice>
    </aixm:Airspace>
```

Note: <<feature>> extensions are treated as a dynamic feature property according to the Temporality Model. This is not the case for <<object>> extensions.

AIXM feature extensions shall be treated as complex properties as defined in section 3.4 of the Temporality Model ("Delta" for complex properties").

Explanation

At the time that extension handling was discussed, there was no business need for treating each property of an AIXM feature extension independently. If one of these properties (e.g. suaType in a sua:AirspaceExtension) changes, then it was assumed that there are good chances that the other properties of the extension (e.g. separationStandard in a sua:AirspaceExtension) change as well (which would be represented in the same delta). A suggestion was made that if the AIXM feature properties in a specific extension model really are independent then they should be included in different namespaces and thus result in different extension elements (e.g. sua1:AirspaceExtension for the suaType property and sua2:AirspaceExtension for the separationStandard property).

The advantage of treating AIXM feature extension properties as complex properties is that the handling of extensions is relatively simple.

A disadvantage of this approach is that the properties added to a feature via a single extension model are not treated as individual properties. In our example, the value of the timeAhead property in an SAA Airspace cannot be changed without also including the valid values for all other SAA Airspace extension properties (saaType, administrativeArea, city and legalDefinitionType) - following the rules defined in section 3.4 of the Temporality Model. This could be solved by treating the properties of an AIXM feature extension class – defined in a single AIXM extension model (like the AIXM SAA extension) – as if they belonged directly to the AIXM feature itself (so ignoring the aixm:extension/myns:AIXMFeature_Extension step in the XML encoding), rather than just being an additional complex property of the feature. A change of the UML to XSD mapping and application schema generation rules does not seem to be required to support this solution.

An AIXM object extension is handled according to the rules defined in section 3.4 of the Temporality Model.

Explanation

<<object>> extensions are different from <<feature>> extensions. <<feature>> extensions have a certain degree of individuality because properties added to an AIXM feature via separate extension models (e.g. the SUA and SAA extension models that extend airspace information) are truly handled as separate feature properties. Their values can be changed individually following the Temporality Model without the need to also include the values for other feature extension properties (added through other extension models) in the delta timeslice that represents the change. In our example, the SUA Airspace extension value can change without the need to repeat the SAA Airspace extension value.

<<object>> extensions, on the other hand, just extend the information contained in the <<object>>. However, an <<object>> is still treated as a complex property and thus always has to include the whole information. If, for example, just the value of the "reservationPhase" property of a SUA AirspaceActivation extension (see Figure 37) changes, then the complete AirspaceActivation has to be updated by including the changed extension value and all unchanged other properties.

The following has not been explicitly covered in the discussion performed on extension handling, but it seems to be a useful observation nonetheless: there is no need to define a specific rule for handling corrections to timeslices that contain extension data because the information in a correction timeslice fully replaces the information in the corrected

timeslice - including the extension data. The data originator needs to ensure that all relevant property values are included in such a correction timeslice.

10.1.6 Temporality Model as Standalone Specification

This section describes reasons for refactoring the AIXM Temporality Model into a standalone standard. It also provides suggestions on which OGC/ISO standards need to be revised/extended to support that goal.

10.1.6.1 Purpose of the Temporality Model

In the aeronautical domain there is a desire to model a feature in a way that allows applications to determine the current and future state of a given feature property. This is relevant for flight planning, flight conduction and in general for aeronautical information management. For a review of past proceedings, the feature history is also relevant. Keeping track of feature property values through time is achieved via the encodings – timeslices as well as properties with schedule – and rules defined by the AIXM Temporality Model (AIXM-TM). The concept of timeslices to express time varying feature properties has its origins in GML itself (see OGC 07-036 section D.3.11).

Due to the way that aeronautical data is managed, different entities may manage different features or feature properties. For example, one authority may assign airspaces with their relevant properties (such as type and extent). Other users like the military then make requests to activate the airspace. As this is an exceptional situation (although expected), the reservation/activation - if granted - results in a temporary change to the otherwise static definition of the airspace. Keeping track of temporary property value changes is supported in the AIXM-TM via the different types of timeslices – in this case *Tempdelta*s. Reservations/activations may also take place based upon a specified schedule – which defines in detail for which times the property has which values. To support such scheduled changes, the AIXM-TM supports the concept of *PropertiesWithSchedule*. This concept can be applied to complex properties. Being able to keep track of possibly frequent AIXM feature property value changes – with such feature properties likely also changing their values independently – is another reason why the AIXM-TM was designed the way it is.

10.1.6.2 Why the Temporality Model should become a Standalone Concept

The aviation domain is pioneering the use of dynamic feature properties in an OGC web service based Service Oriented Architecture. Other domains may benefit from this work as well. In order to not have a strong dependency on a standard defined by the aviation domain the general concepts of the AIXM-TM should be factored out into independent specifications. Having an independent Temporality Model would benefit maintenance/governance of it. Furthermore, all domains that like to express time varying feature properties as well can more easily incorporate a standalone Temporality Model.

Facilitating this re-use of the Temporality Model would also improve interoperability, especially in the way that dynamic features / dynamic feature properties are handled and

managed via (web) services. Improved support by OGC/ISO standards for models that are based on the Temporality Model would also greatly benefit its usability.

10.1.6.3 What is needed?

There are various aspects to consider. On the one hand, current model design and encoding practices need to be extended to support the Temporality Model. On the other hand, standards for managing time varying data need to be extended to better support this type of data. Relevant aspects are described in more detail in the following subsections.

10.1.6.3.1 Model design and encoding

Currently, application schemas that deal with static information only are modeled following the General Feature (meta) Model (see ISO 19109). The model uses stereotyped facets – like the <<FeatureType>>, <<Union>> and <<CodeList>> stereotypes for UML classes – to indicate the specific purpose of each facet. ISO 19103, ISO 19109 and ISO 19136 define a set of basic stereotypes that are used in Application Schema modeling and also define the theory that belongs to these stereotypes. ISO 19136 defines rules for encoding such a model in XML Schema. Each relevant stereotype has its specific encoding rules (see Annex E in ISO 19136). A designer can thus capture the intention of each model facet by choosing the appropriate stereotype for it. Encodings (also called physical models) of the conceptual model can then automatically be generated[9]. The *OWS-8 Domain Modeling Cookbook* (OGC 11-107) provides further guidelines on application schema modeling.

At the moment, AIXM designers just stereotype classes as <<feature>> or <<object>> which governs the encoding. In addition, for <<objects>> that change their value based upon a defined schedule an explicit inheritance relationship to the abstract type *PropertyWithSchedule* is included in the conceptual model.
A designer of a conceptual model that involves time varying properties should be able to explicitly mark which properties are time varying and which are not[10]. This can be achieved via specific stereotypes defined for that purpose. Such stereotypes can also be used to indicate if a type can change its values based on a defined schedule (thus preventing the need to explicitly model the inheritance relationship to *PropertyWithSchedule*). Encoding rules can then be defined to support these new stereotypes. The *OWS-8 AIXM 5.1 Refactoring Report* (OGC 11-106) describes the use of stereotypes and options to support dynamic properties in conceptual models and the encoding into a physical model in more detail.

Having explicit encoding rules would further benefit the modeling work. By stereotyping (feature) types and their properties, the designer can concentrate on the relevant domain aspects – for example on which feature properties always have to be available to applications (even though they may not always be encoded, for example in Tempdeltas).

---

[9] This may require the definition of specific tagged values for the components of the conceptual model.

[10] Note that this is only relevant for feature types, types and data types but not for code lists or unions.

The designer would also be able to more readily document the relevant types (feature type, type, data type), their properties (with intended cardinalities[11]), and relationships between types.

The components of a "normal" application schema can be implemented as usual in object oriented programming. Additional rules would need to be documented for handling application schema that incorporate components with dynamic properties (managed according to the Temporality Model). For example, this includes rules for determining the state of a feature (property) at a given point in time or how to handle cancellation of property value changes.

The following paragraphs outline which revisions / additions are needed in the aforementioned OGC/ISO standards to support the Temporality Model as a standalone concept.

ISO 19103 - Conceptual schema language

Stereotypes commonly used in application schema modeling are defined in ISO 19103. An extension to ISO 19103 should introduce the stereotypes required to generally support time varying properties. Such an extension would not need to be performed by revising ISO 19103 if the standard explicitly allowed that new stereotypes can be introduced by other standards.

ISO 19136 - Geography Markup Language

Rules for encoding time varying properties and other facets of a conceptual model that specifically follow the Temporality Model should be added to ISO 19136. More specifically, there should be an encoding rule for each new stereotype introduced to support the concepts of the Temporality Model in application schema.

Current work on ISO 19136 involves the introduction of a well-defined way for extending the encoding rules defined in ISO 19136. Once this change is available, encoding rules for the new stereotypes can also be documented outside of ISO 19136. This could for example be an extension to ISO 19136.

ISO 19109 – Rules for Application Schema

[11] This may still require that in the encoding a property – if it is time varying – is optional. The cardinality would then have to be checked by the application.

An extension - for example as a "part 2" - should be created for ISO 19109 to introduce the basic theory that is behind the Temporality Model (what are time varying properties, what types of changes exist, etc.) and which would be included in application schema via the new stereotypes. Rules – like those currently contained in the AIXM-TM – can then be documented in detail in an OGC standard. This specification could also include the new encoding rules as mentioned before, thus forming a standalone OGC standard that contains the implementation of the new aspects added to ISO standards to support the Temporality Model as a standalone concept.

Incorporating support for time varying properties in ISO 19109 would provide the foundation for uptake in other standards that deal with features, such as FES and WFS. These two standards are discussed in the following section.

10.1.6.3.2 Managing and accessing time varying data

The Web Feature Service (WFS) provides an interface for managing feature data. The specification is designed to support the current General Feature Model (GFM, defined in ISO 19109). The Filter Encoding Specification (FES) – that is used by the WFS but also other OGC services like the Sensor Observation Service (SOS) and Event Service – defines how filter expressions (possibly involving functions) are defined. The FES was also designed to support the current GFM.

Various issues were encountered and solved – at least to a certain extent, see the reports from previous initiatives and the report *OWS-8 Guidance for Retrieving AIXM 5.1 data via an OGC WFS 2.0* (OGC 11-073) for further details – with accessing and managing AIXM data via the WFS/FES standards. The fact that the XML encoding of a feature no longer follows a simple object/property/value structure but rather an object/timeslice/property/value approach complicated the way that queries need to be written by clients. This is primarily caused by the fact that WFS services to date do not automatically support the business logic for dealing with time varying properties that follow the AIXM-TM. Clients therefore need to deal with the detection of the right timeslices (handling timeslice succession, correction and cancellation) themselves.

Service interface standards like WFS - including standards used by them, like FES – need to be designed to provide simple structures (like operations and functions) to manage and access the type(s) of data they are intended to support. The following paragraphs outline which revisions / additions are needed in the WFS and FES standards to improve the usability of time varying data that follows the Temporality Model.

Filter Encoding Specification

The FES should be extended to support functions for accessing time varying properties in value references without the need to be concerned with the actual encoding via timeslices.

The idea is that value references are still used as usual to identify a certain property in the conceptual model of an application schema. However, just like the function *wfs:valueOf(CharacterString propertyName) : Sequence<Any>* can be used[12] to automatically resolve property values that are (possibly) given by (xlink) reference, specific functions would enable the handling of time varying properties.

Examples of such functions are:

1. *getValues(CharacterString propertyName, TimeInstant time) : Sequence<Any>* – to get the value(s) of a (multi-occurring) property at a given point in time (i.e. past, now or future). This resembles the concept of creating a Snapshot, just that here the value of exactly one property is returned and not the whole feature.
2. *getValues(CharacterString propertyName, TimePeriod time) : Sequence<Sequence<Any>>* – to get the sequence of value(s) of a (multi-occurring) property that are valid during a given period of time. This resembles the concept of Extract described in section 10.1.3, just that here the result is restricted to the sequence of values for one specific property. Note that the result would be a collection of values – depending on whether the property changed its value during the requested period of time or not. In fact, the result is a sequence of sequences to account for situations in which the property is multi-occurring.

If none of these functions was applied – so simply the name of a time varying property is given in a value reference – then the default behavior could be to get the current value of the property, i.e. to use the first function with time=now.

On top of these functions, the FES needs to extend/revise its matching semantics. FES 2.0 already defines the *matchAction* parameter to specify how a comparison predicate shall be evaluated for a collection of values. This is in support of non time-varying but multi-occurring properties. However, the parameter is only defined for some comparison operators (the binary comparison operators). It should be defined for all filter operators (excluding logical operators).

Examples of queries with specific matching requirements:

1. Get features where the value of a specific property equals a given value throughout a given time period – example: determine if a given aerodrome is closed completely in the next two hours. For this we could use the "All" matchAction value defined by FES 2.0. However, as the query involves comparison of a time varying property with a given one it is likely that function two from above would be used, resulting in a sequence of sequences (at least for multi-occurring properties). The semantics for the matchAction may thus need to be revised (to cover this case as well), although the default behavior could be that really all values in the result match the filter (operator).

[12] Unfortunately the function is only defined in WFS, not in FES itself. This requires other standards that use FES to define a similar function. The function should therefore be supported by FES in general.

2. Get features where the value of a specific property equals a given value at least once during a given time period – example: determine if an airspace is going to be active in the next two hours. This matching semantic appears to be covered by the "Any" matchAction value defined by FES 2.0. However, again the semantics of the match action may need to be revised to cover a value reference result that is a sequence of sequences.

More functions and match actions may be of interest for dealing with time varying properties. One should also consider the case in which the given filter operator value itself is time varying. This can for example happen when both values of a binary filter operator reference time varying properties of the model.

A filter encoded in FES 2.0 for the query to determine if an aerodrome is closed completely during a given time period may look as follows:

```
<fes:Filter>
  <fes:PropertyIsEqualTo matchAction="All">
    <fes:ValueReference>aixm:AirportHeliport/fes-
ext:getValues(aixm:availability,2011-09-08T15:00:00+02:00,2011-08-
08T17:00:00+02:00)/aixm:AirportHeliportAvailability/aixm:operationalSta
tus</fes:ValueReference>
    <fes:Literal>CLOSED</fes:Literal>
  </fes:PropertyIsEqualTo>
</fes:Filter>
```

Another example encoded in FES 2.0 to determine if an aerodrome is closed at a given point in time may look like this:

```
<fes:Filter>
  <fes:PropertyIsEqualTo>
    <fes:ValueReference>aixm:AirportHeliport/fes-
ext:getValues(aixm:availability,2011-09-
08T16:00:00+02:00)/aixm:AirportHeliportAvailability/aixm:operationalSta
tus</fes:ValueReference>
    <fes:Literal>CLOSED</fes:Literal>
  </fes:PropertyIsEqualTo>
</fes:Filter>
```

Details of using the proposed functions in value references of filters as well as the semantics of the matchAction parameter need to be discussed in the future. For example, one point for discussion could be how to deal with properties with schedule. However, these examples already show that the usability of querying time varying data based upon the FES standard can be improved significantly if FES provided the according support (either directly or via an extension).

Web Feature Service

WFS implements an interface for managing and querying features. Non-dynamic features, i.e. those that follow the current GFM, can easily be instantiated, updated,

deleted and queried. This should also be the case for features with time varying properties.

The instantiation of a feature with dynamic properties should be straightforward. Feature deletion should be easy as well[13]. Updating and querying time varying data, however, is more complicated. Both actions use the current FES which does not readily support time varying data. To improve the usability of the operations, WFS would need to support an extension/revision of FES as outlined before. The WFS business logic thereby also has to be able to automatically deal with the intricacies of the Temporality Model. This means that the service needs to automatically take timeslice sequencing, correction, cancellation as well as the validTime and schedules into account. Work has been done in the domain of temporal database systems[14] and WFS implementations may leverage this knowledge – or even use built-in support of the database implementation used by the service.

Also note that work on improving the functionality offered by WFS has been performed in previous initiatives and also during OWS-8. See for example the *OWS-8 Guidance for Retrieving AIXM 5.1 data via an OGC WFS 2.0* report (OGC 11-073) for further details.

10.2 SOAP/WSDL support in OWS

Client implementers that were working with one of the WFSs provided for the OWS-8 Aviation thread reported difficulties in binding to that service, as it only supported SOAP based access at that time. While the service now also supports other bindings, the results of the discussion that ensued are worth being documented here, especially because they are not only related to WFS but in general to all web services used in OWS-8 Aviation.

10.2.1 SOAP Complexity

The actual structure of a web service operation request and response is specifically defined for that given type of web service. The specification and documentation of the service contains all information required to implement the relevant operation. For example, the WFS specifies the GetFeature operation request and response structures and relevant operation semantics so that a client can retrieve feature data from the service.

For OWS, the according structures are usually encoded in XML and possibly also KVP encoded requests as well as XML encoded responses. The protocol usually used to interact with another service is HTTP. A service that requires SOAP based communication expects that an operation request is XML encoded and at least wrapped in a SOAP envelope with body element. The additional complexity introduced by wrapping XML encoded requests in a SOAP envelope and unwrapping them again was considered to be unnecessary complexity.

---

[13] Note that the AIXM Temporality Model does not define/use feature deletion operations. AIXM features usually simply run out of "lifetime". For applications that use the envisioned standalone Temporality Model, however, such operations may be relevant.

[14] for example, see http://en.wikipedia.org/wiki/Temporal_database for further details

There was general agreement in the group that such a simple use of SOAP indeed just creates overhead and should be avoided. At the same time, there was consensus that the complexity of dealing with SOAP from an implementation perspective can be mitigated by using available software solutions (also available as open source).

Furthermore, usage of SOAP was accepted to be beneficial when support for common web service functionality - orthogonal to the functionality defined by a specific OWS type - is required. SOAP supports composability of a specific web service standard like WFS or WCS with various WS-* standards such as WS-ReliableMessaging, WS-Security and WS-Addressing. The mentioned standards enable reliable, secure and asynchronous message exchanges between service endpoints in a transport independent and interoperable way. The standards themselves have a varying degree of complexity. Again, existing frameworks and middleware can help reducing the efforts required to implement them. Furthermore, once the required software is in place it can be reused to enable the same functionality when interacting with other web service types.

In theory, SOAP also is a transport independent protocol. So for the rare occasions where HTTP is not available for interacting with a service, an appropriate SOAP binding can be used[15]. However, until recently there was only a standardized SOAP binding for HTTP. Recently, W3C standardized a SOAP JMS binding[16]. Other transports are supported by various toolsets but are not standardized yet.

10.2.2 Bootstrapping a SOAP based OWS

Clients usually start interacting with an unknown OWS by performing a KVP encoded GetCapabilities request sent via HTTP GET to the service. The Capabilities document contained in the response provides information for subsequent interactions, for example about the supported operations, specific endpoints for invoking these operations, allowed operation parameters, supported filter functionality etc.

In the testbed, a WFS initially only supported SOAP based interactions. This required clients to send XML encoded operation requests to the service, wrapped in a SOAP envelope. Information on the correct web service endpoint and SOAP version to use was encoded in a WSDL document. Retrieving this information from the WSDL was deemed unnecessarily complex for such a simple task as retrieving the Capabilities document of an OWS. This relates to the fact that a generic client has to perform a quite detailed investigation of the WSDL document to find out which service endpoint actually realizes the desired operation in case that the WSDL lists more than one service element, which appears to be valid.

One way to approach the problem of bootstrapping OWSs that support different bindings is described in the following:

---

[15]Although HTTP is ubiquitous, it may not always be feasible to use it due to domain and/or application specific requirements.

[16]Note that JMS is an API and does not define full end to end service interoperability.

- Whenever an OGC Web Service supports KVP encoded requests via HTTP GET and/or plain XML encoded requests via HTTP POST, the use of the well known GetCapabilities request can be used for bootstrapping because all the required information is contained in the Capabilities document. For each network endpoint indicated by a URL, the use of either the HTTP GET or POST method is given, in addition to other parameters as defined by OWS Common
- However, if SOAP is required then the Capabilities document does not carry enough information. The use of SOAP may involve a particular format to include security related metadata into the headers. Different OASIS standards such as WS-Policy or WS-SecurityPolicy exist that allow encoding constraints to indicate what the service expects in terms of security metadata when a clients tries to execute it. Other information, such as the required use of HTTP headers such as *SOAPAction*, cannot be specified in the Capabilities document. Therefore, the bootstrapping of an OWS that requires SOAP should be done by a WSDL document that describes in more detail the requirements to "get going" with the service.
- In terms of publish-find-bind, it is up to the service provider to publish the appropriate bootstrapping mechanism in a registry or catalogue. For HTTP GET and POST OWS this should be the Capabilities document and for SOAP OWS it should be the WSDL document. Both can be provided via a simple online document and do not require the client to perform an additional service request. This relaxes the opportunities for bootstrapping a protected OWS. Access to the Capabilities or the WSDL document can always be unprotected. In case that the service is protected and the user does not have the required rights, he will find out after the first service execution attempt.

In the end, the group agreed to not use SOAP during the testbed for WFS interactions as the additional features that it could have supported were not required there. The WFS service that initially only supported SOAP was enhanced to also support KVP and XML based operation invocations.

10.2.3 SOAP Version

When talking about SOAP and support for it by a given web service, it is important to acknowledge that there are two versions of SOAP: 1.1 and 1.2. Communities may require support for SOAP 1.1 or 1.2 or both or any one of them while allowing the other. There may be different reasons for doing so, such as backwards compatibility or integration with existing systems. Therefore, OWS standards should be SOAP version agnostic even though usually more options are not favorable for achieving interoperability.

The Sensor Web Enablement 2.0 standards are written in a SOAP version agnostic way. They rely on OWS Common 1.1, which does not specify any requirement regarding SOAP. In addition, they do not have any requirement that prevents a SWE service from using a particular SOAP version. Other OWS standards used in the OWS-8 Aviation thread are not SOAP version agnostic:

- WCS 2.0 builds upon OWS Common 2.0, which requires SOAP 1.2.

- WFS 2.0 builds on OWS Common 1.1 which - as said before - does not define SOAP in any way, but the WFS 2.0 specification itself requires SOAP 1.1.

Apparently some harmonization is needed. Until that is achieved, aviation clients may need to support both versions of SOAP in a SOAP based service environment.

10.2.4 Security

The use of SOAP - as a particular format for XML messages exchanged with a network endpoint - provides hooks for enabling metadata with the actual information exchanged in an interoperable way. One useful way is to use these hooks for including message based security metadata to cope with Integrity, Confidentiality and potentially Authenticity for the XML encoded information (enclosed by the SOAP message tags).

This is important in architectures where there is no direct connection between the service itself and the client that executes the service, because lower ISO/OSI stack security measures such as SSL or TLS only provide a secure communication channel between two machines and not between the actual applications running on it.

Therefore, in a workflow or a real message based system such as an Enterprise Service Bus (e.g. SWIM) it is important to provide a method that enables securing the exchanged information (the message) from one application to another (end-to-end). This is achieved by explicitly encrypting a message before it is delivered via the network to the recipient.

However, in an architecture where a Web Browser or a desktop client executes one or multiple services according to the transparent chaining pattern (OGC Topic 12), the use of SOAP does not seem to bring any advantages compared to using an HTTPS (HTTP + SSL/TLS) connection, because the difference between point-to-point and end-to-end is not meaningful here. It actually tends to be more complex for application and service developers, because the use of secured SOAP messages requires to encrypt and decrypt each individual request/response inside the application compared to simply using the security features that are already provided and automatically handled by the transport protocol (HTTPS).

As encryption based security was not required in OWS-8, the use of SOAP was not necessary. This was also beneficial for the access control use cases, as the PEP did not need to perform addition processing to unwrap the actual request contained in a SOAP message.

10.2.5 Message Size

Implementers reported a doubling of size of exchanged web service messages when SOAP was used - for example for GetCapabilities requests. Some observations regarding the statement are listed in the following:

- A very basic GetCapabilities request encoded in XML can be very small. The same can be true for other OWS operations. In these cases simply wrapping the request with a SOAP envelope and body element indeed represents overhead that

is seldom necessary. However, the structure of these requests allows for more detailed parameterization of the request, thus essentially increasing the size of the core request. In such a case, the ratio of core request size to SOAP wrapper size changes. The size of a complex request is often multiple times bigger than the SOAP envelope that surrounds it.

- The application of compression techniques that were investigated in the testbed can significantly decrease the size of a request, so that the size increase of a SOAP request is diminished.
- As outlined before, SOAP enables common web service functionality that is orthogonal to the core operations of the given OWS. The advantage of being able to support this functionality in a common way for any OWS operation appears to outweigh the disadvantage of potentially doubling the request size.

10.2.6 Conclusion

Using SOAP based OWS interactions is more complex than using a KVP and/or plain XML based operation encoding. At the same time, SOAP can be useful to avoid re-inventing the wheel to enable common web service functionality such as secure and reliable message exchanges. Service infrastructures may not have a need for such functionality. If that is the case, SOAP should not be used.

10.3 Unit of Measure Handling in Filter Expressions

10.3.1 Background

Clients often query AIXM features by filtering on feature properties that are quantities, i.e. which have a value provided in a certain unit of measure (UoM). For example, the length of a runway can be stated in meter, kilometer but also in foot or yards.

The numerical value of runway length changes based upon UoM it is provided in. Filtering AIXM data based upon simple value comparison of quantities against given numbers therefore only makes sense if the client knows the “native” UoM of the property that the comparison operator is applied upon. In that case, the client can ensure that the comparison value provided in the query is given in the same UoM. However, if the UoM for a given feature property is not defined as a constant in a GML Application Schema (such as AIXM) then clients would need to determine the actual UoM of a given AIXM feature property first before being able to perform a meaningful query that involves a comparison operator on this property.

A much more convenient solution for this issue would be to have the queried service automatically convert quantities - both in feature properties and ad-hoc queries with filter expressions contained in client requests - into their base units and then perform comparison operators. Such a conversion is of course not necessary if the UoM of both values is the same.

Example:

If the length of a specific runway was provided in foot then the runway length value could be converted to meters by simply multiplying the original value with 0.30479976 (a foot being 12 inches and an inch being 0.02539998 m as defined by UCUM). Likewise, a quantity given as comparison value may be given in meters. Let us say that a client is interested in finding airports with runways with a length bigger than or equal to 3000 meters (e.g. for landing). If a given runway had a length of 8000 feet then the naive number comparison without UoM conversion would result in a match, although the runway is in fact shorter than 2500 meters. With automatic UoM conversion, the service would know that runway length had to be bigger than or equal to ~9842.53 feet to produce a match.

10.3.2 Enabling automated Unit of Measure Conversion

Querying features at a WFS is performed via a GetFeature request which supports query elements that include filter expressions according to the OGC Filter Encoding Specification (FES). Comparison operators in FES filter expressions reference the feature property that the operator shall be applied to and either provide the comparison value as literal directly or reference it. The following paragraphs discuss how automatic UoM conversion can be enabled at a WFS[17].

First of all, we need to identify how a service can know the UoM that a feature property and comparison value is given in. There are different options:

- UoM is constant - this would be defined in the GML Application Schema of a given feature type; note that in case that the comparison value is given by reference, the service would need to determine the feature type that the referenced property belongs to
- UoM is known to be given in base units - this would be a requirement for the given domain, but is probably hard to enforce; in addition, the domain would need to agree on the base units (though following the definition of base units provided by UCUM appears to be beneficial)
- UoM is explicitly provided - either in the dataset (e.g. as additional column in the database of the service) or in the filter expression.

Explicitly providing the UoM of a comparison value in a filter expression was discussed during the testbed. The participants agreed that explicitly adding the UoM information to the literal representation of the comparison value is the best option.

---

[17] WFS 2.0 explicitly states that it "*does not define any support for handling conversions between unit of measure*" - see section 7.9.2.5.3.5 "Units of measure handling" in OGC 09-025r1 / ISO/DIS 19142

```
<fes:PropertyIsGreaterThanOrEqualTo>

<fes:ValueReference>aixm:Runway/aixm:timeSlice/*/aixm:lengthStrip</fes:
ValueReference>
  <fes:Literal>
    <gml:measure uom="m">3000</gml:measure>
  </fes:Literal>
</fes:PropertyIsGreaterThanOrEqualTo>
```

Note 1: the Sensor Event Service (OGC 08-133) introduced a similar way to perform UoM conversion. The difference there was that the comparison literal includes an element of the GML ScalarValue group, i.e. a gml:Boolean, gml:Count, gml:Category or gml:Quantity - the latter of which is of type gml:MeasureType and thus has the uom attribute attached to it. This approach allows even more explicit indication of the type that the Literal is given in. The “type” attribute that can be added to an fes:Literal could be used to indicate the type of the fes:Literal content. This is useful for primitive content, such as a boolean value (in which case the type attribute could be set to “xs:boolean”) but not so useful for element content where the element usually follows a global element definition.

Note 2: The discussion how to represent the UoM of a comparison value also included discussion of the case that the UoM is given in. It was suggested to set the “matchCase” parameter of a comparison operator to false in order to ensure that the request is fulfilled regardless of whether the UoM is defined as, for example, ‘M' or 'm'. However, discussion of this revealed issues with this approach. Sticking to the example where meter is the UoM, "Mm" (megameter) and "mm" (millimeter) both use case sensitive UCUM symbols ("s" second vs. "S" Siemens is another example without prefixes). The same with case insensitive symbols (see UCUM for further details) would be "MAM" and "MM" (with "mAm", "mam", "Mam" etc and "mM", "Mm", "mm" being equivalents because of case insensitivity). Ignoring case would be dangerous when case sensitive symbols are used - one would compare otherwise incompatible uoms or confuse their actual value (by misinterpreting the prefix). If a domain chooses to use only case sensitive uoms, then this needs to be clearly documented. GML does not seem to restrict case sensitivity on the UomSymbol type. As UCUM states, the use of case insensitive symbols may be the greatest common denominator.

Now that the various ways to provide the UoM for feature property and comparison value have been discussed, we need to identify the different situations in which UoM conversion can or cannot be performed:

UoM in feature property known (either constant, explicitly provided or known to be given in base units)	**UoM in comparison value known (either constant [if known for the feature type of the referenced value], explicitly provided or known to be given in base units)**	**UoM conversion possible**
yes	yes	yes

yes	no	no
no	yes	no
no	no	no

Automatic UoM conversion can only be performed if the UoM for both the feature property value and the comparison value is known. Otherwise, the service would need to resort to perform direct number comparison.

10.3.3 Conclusion

Automated UoM conversion as described in this section would improve the filter capabilities of OGC Web Services. Especially when the UoM of a feature property is not constant, the mechanism helps to perform meaningful comparison operations.

Eventually, the mechanism should be integrated in or become an extension of the OGC Filter Encoding Specification.

The discussion performed in OWS-8 on this topic did not cover all relevant aspects. A proper specification of automated UoM conversion would need to define:

- how to identify UoMs (of both the feature property targeted by a comparison operator and the comparison value itself) - this has been discussed in OWS-8 (see previous section)
- the behavior in case that UoM conversion is not possible - this was not discussed during the testbed
- how to advertise that the service supports UoM conversion and which UoM definitions it understands - this was also not discussed during the testbed, but the SES (OGC 08-133) has some information on this that may be useful (essentially, the filter capabilities are extended to indicate UoM conversion capabilities)
- how to handle precision errors in comparison operations caused by value conversion - rounding errors may be introduced through the conversion process, which may or may not be relevant for a given application; this was not discussed during the testbed
- how case sensitivity of UoM symbols is handled - this has been discussed to a certain extent (see previous section)

11 Scenarios

Several realistic scenarios were developed for the OWS-8 Aviation thread. The demonstration of the OWS-8 Aviation thread was inspired by these scenarios – detailed scenario information is provided in chapter 0.

The scenarios provided a fictitious but realistic context for testing the developed functionality. They prompted the exercising of interfaces, components, tools and services as well as the use of encodings. This includes exercising a variety of web services

utilizing AIXM and WXXM encodings (including digital NOTAMs, information about field conditions and relevant meteorological information).

12 Accomplishments

The OWS-8 Aviation thread demonstrated the successful coordination and cooperation of 20 diverse organizations in testing and advancing OGC, Aviation and other standards in a rapid prototyping environment. As such, the major accomplishments of OWS-8 Aviation include:

- A basic Authoritative Data Source Architecture was developed. It provides the foundation for ensuring integrity and confidentiality of aeronautical data in a service oriented architecture based on OGC standards. – For further details, see OGC 11-086.

- Practical guidelines to domain modeling following a series of best practices were documented and applied towards improving the efficiency and reusability of the AIXM model. – For further details, see OGC 11-107.

- Suggestions were developed and documented for refactoring AIXM (and more specifically the Digital NOTAM Event model) to improve the overall model quality and to better align the AIXM model with the OGC/ISO standards baseline. – For further details, see OGC 11-106.

- Recommendations for improving the overall support of time varying feature data (following the AIXM Temporality Model) within the OGC/ISO standards baseline were identified. The recommendations are intended to considerably improve the usability of time varying data with respect to model design and encoding as well as data management and access – not only in the Aviation domain but in all domains that have a need to deal with time varying data. – For further details, see 10.1.6 in this document as well as OGC 11-106.

- Suggestions for enhancing the AIXM Temporality Model but also for solving identified issues were developed and documented. – For further details, see section 10.1 in this document.

- Initial guidance for configuring and using a WFS 2.0 for managing and serving AIXM data have been drafted. Such guidance is intended to support consistent implementation of WFS 2.0 in the Aviation community. – For further details, see 11-073.

- The suitability of OGC standards for portraying aeronautical data with ICAO symbology was investigated. Issues were identified and recommendations for solving them were provided in the form of change requests to OGC standards. – For further details, see OGC 11-089.

- A number of compression algorithms were investigated. All of them provide a good overall performance for compression of AIXM, allowing the usage of AIXM over data link connections. – For further details, see OGC 11-097.

- The Event Architecture developed in previous OGC initiatives was further advanced to support the accurate delivery of digital NOTAMs. Three new features and concepts were developed and tested: event enrichment, dynamic filtering and pull support. In addition, OWS-8 is the first OGC initiative where two independent Event Service implementations were available which benefited interoperability testing considerably. – For further details, see chapter 9 in this document.

- OWS-8 participants performed a detailed review of the Digital NOTAM Event Specification version 1.0. The specification was validated through both a thorough conceptual review as well as the implementation of a suite of executable schematron tests. Identified issues were documented and recommendations for addressing them in the next version of the specification were provided. – For further details, see OGC 11-092.

- Work was performed on advancing the use of WXXM and Weather Concepts in the Aviation domain. The applicability and suitability of WXXM in providing accurate weather data and serving it via OGC Web Coverage Services was investigated and demonstrated. – For further details, see OGC 11-072.

- An audit of the WXXM XML Schema was performed, revealing a number of issues regarding the compliancy with encoding rules defined in ISO 19136. The audit results will be useful in revising the WXXM schema to improve compliancy with the OGC/ISO standards baseline. – For further details see OGC 11-091.

- Guidelines on using ISO metadata in AIXM 5.1 were further documented, in compliance with the requirements and recommendations on metadata within the Aviation domain. – For further details, see OGC 11-061.

13 Annex A – Detailed Scenario Descriptions

This Annex provides detailed information on the scenarios developed for the OWS-8 Aviation thread. The scenarios revolve around the following themes:

- Continued support for dispatch and planning activities,
- Increasing situational awareness,
- Demonstration of the use of probabilistic information in weather data in decision-making applications.

13.1 Dispatch and Planning

This scenario is based upon the OWS-7 Dispatch scenario[18]. That scenario was developed to demonstrate the use of OGC web services in providing flight dispatchers and pilots with an alternative source for much of the information that is needed in the flight planning, pre-flight briefing and flight following processes. The underlying transmission methods for web services were assumed to exist. Participants in these processes are flight crew, ground controllers, custodians and providers of aeronautical information (and information updates), and custodians and providers of weather information. The role of ATC was de-emphasized in the scenario, but it may be assumed that all in-flight operations are carried out in concert with ATC authorities.

Web services were used to deliver aeronautical information encoded in AIXM and weather information encoded in WXXM to flight dispatcher workstations and pilots portable devices. The flight dispatcher retrieves aeronautical data and weather data pertinent to the planned routes of proposed flights when preparing flight-briefing packages. Shortly prior to a flight or when the pilot is at the departure gate or in the cockpit, the pilot could download the flight-briefing package to his Electronic Flight Bag (EFB) using web services. The pilot could also use web services enabled in his EFB to update the aeronautical and weather information in the briefing package or to obtain additional information, e.g. about features that are not covered or are insufficiently covered in the briefing.

The following list describes the sequence of steps of the Tsunami scenario developed for OWS-8. Selected steps of this scenario were demonstrated in the final OWS-8 demonstration.

1. The flight dispatcher checks into work and finds that he is responsible for providing preflight briefing packages and flight following services for a flight from Tallinn (IATA code: TLL) to Honolulu (IATA code: HNL) via Chicago (IATA code: ORD) due to depart within the next 8 hours.
2. The dispatcher retrieves information relevant for the planned flight:

[18] More information on the scenario can be found in the OWS-7 RFQ Annex B, section 4.4.5 (http://portal.opengeospatial.org/files/?artifact_id=36132&format=pdf).

   a. Weather information: SIGMETs (icing, turbulence, etc.) along the flight route as well as METARs and TAFs (wind, precipitation, visibility etc) for destination and possible diversion and alternate airports.
   b. Airport information: information for identification of diversion and alternate airports (available passenger terminal, re-fueling facilities, hard-surface runway of certain required minimum runway length etc).
   c. Additional aeronautical information: for example airspace activation information along the flight route.
3. The dispatcher visualizes and inspects all information on its client to check the planned flight.
4. The dispatcher selects a flight route as well as diversion and alternate airports.
5. The dispatcher subscribes to be automatically notified of any changes in weather conditions or aeronautical features relevant for the flight (such as airports and airspaces).
6. The dispatcher puts all relevant information into the preflight briefing package and sends it to the pilot before takeoff.
7. The flight takes off as planned.
8. While the flight is over the United States, the dispatcher receives several SAA activation NOTAMs. The dispatcher visualizes the new data and determines that the activations do not affect the flight.
9. An earthquake occurs near Japan, causing a tsunami that approaches Hawaii.
10. A digital NOTAM informs the dispatcher about the total aerodrome closure of the alternate airport (the alternate airport is not operational).
11. The dispatcher searches for another alternate airport (checking weather and aeronautical information for potential candidates), finds one and changes the flight plan to use the new alternate airport.
12. A digital NOTAM informs the dispatcher about a runway closure at the destination airport.
13. The dispatcher displays the runway information for Honolulu airport and determines that another runway is still available and can be used for landing. The dispatcher updates the flight plan accordingly.
14. Plane lands successfully in Honolulu.

13.2 Increasing Situational Awareness for Flight Planners, Pilots and Operation Centers

Another scenario was developed to demonstrate how application clients can use real-time weather information to enable an increased level of situational awareness for flights planners, pilots and operation centers. The data is gathered by the NASA Global Hawk UAS on equatorial flights of extended (30 hour) duration and made accessible via OGC Web Services. Unfortunately, delays in getting sample data resulted in this scenario not being exercised or demonstrated in OWS-8.

13.3 Probabilistic Weather in Decision Making

The use and visualization of Probabilistic TAFs in application clients was exercised and demonstrated in OWS-8. The demonstrations were inspired by the following scenario

using probabilistic weather information in flight planning operations of a parcel shipping company.

A parcel shipping company uses a fleet of Boeing 727-200 aircraft daily to deliver its packages. Using a "hub-and-spoke" sort and distribution system, meteorologists must forecast weather conditions at several airport hubs at critical early morning hours, 14 to 16 hours in advance. Cost/Loss analysis indicates that it is better for the company to increase staffing and send flights to alternate hubs when the risk of LIFR condition at the primary hub is 40% or greater.

On 26 January 2011, at the primary hub airport, KBNA, Nashville, TN, a stationary front lies just to the south. MVFR and occasional IFR conditions at the airport are expected to persist for 24 to 36 hours. Precipitation is expected within the next 12 to 18 hours. Current TAF for the primary hub airport has a PROB group of IFR ceilings and rain/snow during the overnight hours. LAMP guidance for the same time period indicates high probability of IFR conditions and snow with possibility of freezing precipitation as well. As each hour passes with each LAMP guidance update, and as the critical decision period approaches, LIFR and freezing precipitation probabilities at the hub airport steadily increases for the nighttime period. At decision time, probability of LIFR condition at the hub airport reaches 37% with 44% chance of freezing rain and/or sleet. The dispatcher, examining the current LAMP probability trends, decides to send more than half its planes to alternate hubs.

Probabilistic weather information is also important for pilots. The following list briefly describes three possible scenarios.

1. Vacation on 26 August 2010, initial flight plan: Gulfstream JetProp Commander 980 @ 25kft from KLAX to KJAC, alternate KDIJ. Departure 9:30 LT. Flight time 3:30 h.
    a. Get latest TAF for destination airport and alternate (issued at 11:20Z). Notes thunderstorms forecasted for KJAC and KDIJ in afternoon.
    b. Query and get the latest LAMP probabilities of thunderstorms & IFR conditions at arrival time (16Z-18Z) at KJAC. Based on latest guidance available, decides to proceed and files plan for KJAC.
2. Business meeting on 15 August 2009, private plane, initial flight plan: Cessna 172S @ 10kft flying VFR from KBWI to KCRW. Departure 09:30Z. Flight time 1:20 h.
    a. Retrieves latest official TAF for KCRW (issued 05:34Z). Notes MVFR conditions (5SM) at KCRW due to light fog (BR) at 11Z (arrival time) with improving conditions thereafter.
    b. At 09Z, pilot queries the latest LAMP probabilities of MVRF, IFR conditions at arrival time (10Z to 12Z) and decides the IFR probabilities are too high and delays departure.
    c. Shortly after 09Z, METAR SPECI observations indicate the development of significant restriction in visibility due to rapid development of fog. The official TAF for KCRW is amended at 09:20Z.

   d. At 10Z and 11Z LAMP guidance continues to indicate high IFR probabilities for the next few hours. A new KCRW TAF issued at 11:20Z indicates that the 1SM BR condition will dissipate by 14Z at KCRW. Pilot schedules departure at 13:30Z

3. Winter vacation beginning 26 February 2011, private plane, initial flight plan: Beechcraft KingAir B100 @ 24kft from KLVS to KSUN. Departure 15:30Z. Flight time 3:45 h.
   a. Gets latest TAF for KSUN. Pilot notes IFR conditions at KSUN due to ceilings beginning at 20:00Z. Queries LAMP probabilities for Low IFR at KSUN for which aircraft is not equipped. LIFR probabilities are low, also notes that LAMP guidance is forecasting at least 23 knot gusts beginning in the afternoon, much of it direct crosswind (Rwy 13/31), which are not in the official TAF. Pilot decides to depart for KSUN.
   b. While en route, pilot subscribes for observations and updates to TAF and LAMP guidance for KSUN. As each hour passes, updated LAMP guidance increases the probability of Low IFR conditions, and wind speeds and gusts expected at KSUN around arrival time (19Z-20Z).
   c. One hour and fifteen minutes prior to arrival (17:54Z METAR), light snow begins to fall at KSUN with 33kt gusts. Pilot files updated flight plan indicating KPIH as an alternate. Thirty-five minutes before arrival, KSUN TAF is updated indicating IFR conditions both in ceiling and visibility and strong winds, occasional LIFR condition due to snow reducing visibility and ceilings. Pilot decides to fly to alternate destination, KPIH which remains MVFR.